НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
імені ІГОРЯ СІКОРСЬКОГО»

Факультет прикладної математики

Кафедра програмного забезпечення комп'ютерних систем

«На правах рукопису»
УДК 004.4'24

«До захисту допущено»

Науковий керівник кафедри

_____ Іван ДИЧКА

«___»_____2021 р.

Магістерська дисертація

на здобуття ступеня магістра

за освітньо-науковою програмою
«Інженерія програмного забезпечення комп'ютерних та інформаційно-пошукових систем»

зі спеціальності 121 Інженерія програмного забезпечення

на тему: «Метод побудови моделі адаптивного асоціативного кешу в інформаційних системах»

Виконав:
студент ІІ курсу, групи КП-91мн
Охримчук Денис Дмитрович _____

Керівник:
Доцент кафедри ПЗКС, к.т.н.,
Люшенко Леся Анатоліївна _____

Консультант з нормоконтролю:
Доцент кафедри ПЗКС, к.т.н., доцент
Онай Микола Володимирович _____

Рецензент:
Доцент кафедри ММСА ІПСА, к.ф.-м.н., доцент,
Шубенкова Ірина Анатоліївна _____

Засвідчую, що у цій магістерській дисертації немає запозичень з праць інших авторів без відповідних посилань.
Студент _____

Київ – 2021 року

Національний технічний університет України
«Київський політехнічний інститут імені Ігоря Сікорського»
Факультет прикладної математики
Кафедра програмного забезпечення комп'ютерних систем

Рівень вищої освіти – другий (магістерський)

Спеціальність – 121 «Інженерія програмного забезпечення»

Освітньо-наукова програма «Інженерія програмного забезпечення комп'ютерних та інформаційно-пошукових систем»

ЗАТВЕРДЖУЮ
Науковий керівник кафедри
_____ Іван ДИЧКА
(підпис)
«__» _____2019 р.

ЗАВДАННЯ
на магістерську дисертацію
студенту Охримчуку Денису Дмитровичу

1. Тема дисертації «Метод побудови моделі адаптивного асоціативного кешу в інформаційних системах» науковий керівник дисертації Люшенко Леся Анатоліївна, к.т.н., доцент, затверджена наказом по університету від «26» березня 2021 р. № 899-С.
2. Термін подання студентом дисертації «18» травня 2021 р.
3. Об'єкт дослідження: процес створення та аналізу методів кешування.
4. Предмет дослідження: методи роботи кешу, адаптивні алгоритми, алгоритми побудови асоціативних правил.
5. Перелік завдань, які мають бути вирішені:
   - провести аналіз існуючих методів кешування в інформаційних системах, виявити їх переваги та недоліки;
   - запропонувати новий метод, що покращує процес кешування у порівнянні із існуючими методами;
   - створити програмне забезпечення, яке реалізує запропонований метод;
   - провести дослідження запропонованого методу та порівняти його з існуючими, проаналізувати отримані результати;
   - оформити звіт з роботи у вигляді документації магістерської дисертації та публікації в науковому виданні.
6. Перелік ілюстративного матеріалу:
   - алгоритм кластеризації, що враховує статистичні співвідношення;
   - алгоритм побудови та оцінки моделі;
   - використання одного і декількох наближень в різних станах;
   - діаграма моделі програмного забезпечення;
   - схема етапів методу;
   - однозначний поділ простору.

7. Перелік публікацій:

- Тези доповіді «Адаптивний асоціативний кеш в багатоагентних високоінтегрованих інформаційних системах».
- Стаття «Концепція цілочислової норми оцінки різниці між елементами зображення з врахуванням балансу білого».

8. Консультанти розділів дисертації

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		Завдання видав	Завдання прийняв
Нормоконтроль	Онай М.В., доцент кафедри ПЗКС		

9. Дата видачі завдання «15» жовтня 2019 р.

Календарний план

№ з/п	Назва етапів виконання магістерської дисертації	Термін виконання етапів дисертації	Примітка
1.	Формулювання мети дослідження та завдання на магістерську дисертацію, ознайомлення з тематикою роботи	15.01.2020	
2.	Підбір та вивчення літератури, наукових матеріалів; визначення структури магістерської дисертації;	25.05.2020	
3.	Проведення наукового дослідження; робота над першим розділом дисертації	01.10.2020	
4.	Проведення наукового дослідження; робота над другим розділом дисертації; підготовка тез для доповіді на ПМК-2020	25.12.2020	
5.	Проведення наукового дослідження; розроблення програмного забезпечення	10.01.2021	
6.	Проведення наукового дослідження; робота над третім розділом дисертації;	10.02.2021	
7.	Завершення роботи над основною частиною дисертації; науково-дослідна практика;	20.03.2021	
8.	Оформлення текстової і графічної частини магістерської дисертації	12.05.2021	

Науковий керівник _____ Леся ЛЮШЕНКО

Студент _____ Денис ОХРИМЧУК

РЕФЕРАТ

Актуальність теми. Сьогодні існує потреба в прискоренні роботи програмних інформаційних систем. Одним з інструментів підвищення бистродії виконання запитів є кешування.

Кешування має змогу покращити швидкодію роботи програмного забезпечення, яке потребує значної кількості обчислювальних ресурсів. Загальним недоліком існуючих методів кешування є недостатня ступінь адаптованості до роботи інформаційних систем, в рамках яких застосовуються існуючі методи кешування. Недоліки існуючих методів кешування можуть бути подолані шляхом розроблення методу кешування, а саме – адаптивного асоціативного кешу. Дана робота присвячена розробці та дослідженню метода адаптивного кешування даних інформаційної системи з використанням асоціативних правил, які будуються на основі використання статистичних даних щодо роботі системи.

Об'єктом дослідження є процес створення та аналізу методів кешування.

Предметом дослідження є методи роботи кешу, адаптивні алгоритми, алгоритми побудови асоціативних правил.

Мета роботи полягає в підвищенні ефективності роботи систем кешування в інформаційних системах шляхом розроблення адаптивного асоціативного модулю кешування.

Методи дослідження. В даному дослідженні використовуються методи моделювання систем, аналіз обробки статистичних даних, розробка та тестування нових методів.

Наукова новизна полягає в тому, що був створений новий метод кешування. Особливостями розробленого методу кешування є адаптація системи кешування відповідно до асоціативних правил, які будуються на основі статистичних даних щодо використання інформаційної системи. Таким чином кешування інформаційної системи адаптується до

попереднього використання інформаційної системи. При наявності набору таких асоціативних правил, інформаційна система буде мати можливість розміщувати результати виконання ряду запитів в кеш ще до їх виникнення. Це надає можливості системі покращити показники швидкодії надання результатів обробки запитів.

Практичне значення отриманих результатів полягає в тому, що запропонований метод кешування дозволяє досягти кращої ефективності та швидкості взаємодії із інформаційною системою шляхом покращення показників кількості влучень у кеш.

Апробація роботи. Основні положення та результати роботи були представлені та обговорювались на науковій конференції магістрантів та аспірантів «Прикладна математика та комп'ютинг» ПМК-2020 (Київ, 18-20 листопада 2020 р.) та опубліковані у збірнику «Управляючі системи і машини».

Структура та обсяг роботи. Магістерська дисертація складається зі вступу, чотирьох розділів, висновків та додатків.

У вступі коротко описана проблематика роботи, обґрунтовується її актуальність, наводиться оцінка сучасного стану досліджуваної області.

У першому розділі детально розглядається область дослідження, наводяться існуючі методи кешування, виявляються їх переваги, недоліки та можливості для покращення.

Другий розділ присвячено створенню адаптивного асоціативного методу кешування. Наводиться опис створеного методу кешування та архітектури цільової інформаційної системи.

У третьому розділі наводиться розроблене програмне забезпечення, яке реалізує метод адаптивного асоціативного кешування та інформаційну систему для його тестування. Виявлено та описаного технології розробки програмного забезпечення, обрано архітектуру ПЗ, спроєктовано його компоненти.

Четвертий розділ присвячено опису результатів застосування адаптивного асоціативного кешу в рамках розробленої моделі інформаційної системи. Проведено тестування та порівняння адаптивного асоціативного кешу із декількома іншими методами кешування.

У висновках наводиться короткий підсумок роботи.

У додатках наводяться копії ілюстративних матеріалів та коду створеного програмного забезпечення.

Робота виконана на 78 аркушах, містить 3 додатка, посилання на список використаних літературних джерел з 27 найменування. У роботі наведено 13 рисунків та 11 таблиць.

Ключові слова: інформаційна система, асоціативні правила, метод кешування, ефективність кешу.

ABSTRACT

Actuality. Today there is a need to accelerate the work of software information systems. One of the tools to increase the speed of query execution is caching.

Caching can improve the performance of software that requires a significant amount of computing resources. A common disadvantage of existing caching methods is the lack of adaptability to the operation of information systems, which use existing caching methods. The disadvantages of existing caching methods can be overcome by developing a caching method, namely – adaptive associative cache. This work is devoted to the development and research of the method of adaptive data caching of the information system using associative rules, which are based on the use of statistical data on the system.

Object of research is is a process of creating and analyzing caching methods.

Subjects of research are methods of cache operation, adaptive algorithms, algorithms for constructing associative rules.

Goal of the work is to increase the efficiency of caching systems in information systems by developing an adaptive associative caching module.

Methods of research include methods of modeling systems, analysis of statistical data processing, development and testing of new methods.

Scientific novelty of the work is that a new method of caching has been created. The peculiarities of the developed caching method are the adaptation of the caching system in accordance with the associative rules, which are based on statistical data on the use of the information system. Thus, the caching of the information system is adapted to the previous use of the information system. With a set of such associative rules, the information system will be able to place the results of several queries in the cache before they occur. This allows the system to improve the performance of query results.

Practical value of the received results is that the proposed method of caching allows to achieve better efficiency and speed of interaction with the information system by improving the number of hits in the cache.

Approbation. The main provisions and results of the work were presented and discussed at the scientific conference of undergraduates and graduate students "Applied Mathematics and Computing" PMK-2020 (Kyiv, November 18-20, 2020) and published in the collection "Control Systems and Machines".

Structure and content of the thesis. Master's thesis consists of an introduction, four chapters, conclusions and appendices.

The introduction provides a general description of the work, evaluated the current state of the problem, substantiated the relevance of the research direction, formulated the purpose and objectives of the study.

The first section examines in detail the area of research, provides existing methods of caching, identifies their advantages, disadvantages and opportunities for improvement.

The second section is devoted to creating an adaptive associative caching method. The description of the created method of caching and architecture of the target information system is given.

The third section presents the developed software that implements the method of adaptive associative caching and information system for its testing. The software development technology is identified and described, the software architecture is chosen, its components are designed.

The fourth section is devoted to the description of the results of the application of the adaptive associative cache within the developed model of the information system. The adaptive associative cache was tested and compared with several other caching methods.

The conclusion contains brief overview of the results obtained in the work.

The appendices provide copies of illustrative materials and code of the created software.

Keywords: information system, associative rules, caching method, cache efficiency.

ЗМІСТ

СПИСОК ТЕРМІНІВ, СКОРОЧЕНЬ ТА ПОЗНАЧЕНЬ

Влучення у кеш – запит інформації з інформаційної системи яка є всередині кешу.

Ефективність кешу – відношення кількості влучень до кешу до загальної кількості запитів.

Cache miss – відсутність влучення у кеш під час запиту.

Інформаційна система – система, що надає інформацію у відповідь на запит до інтерфейсу інформаційної системи.

СУБД – система управління базами даних

Системи управління базами даних – це системи, що призначені для зберігання та маніпулювання даними.

Інтерфейс – це частина системи, що надає можливості взаємодії та обміну даних з системою.

Архітектура системи – це сукупність зв'язків між частинами системи. В стандарті ISO/IEC/IEEE 42010:2011 “Systems and software engineering – Architecture description” архітектура – це властивості системи, що втілені в її елементах, зв'язках між ними і принципах проєктування [1].

Скрипт – послідовність дій, що написана розробником для виконання середовищем, у якому запускається. Може бути представлений окремим файлом, який використовує програма для виконання дій, записаних у файлі на скриптовій мові.

Скриптова мова програмування – мова програмування, що орієнтується на написання скриптів що потім використовуються у роботі програмних застосунків.

Оперативна пам’ять – швидкодійна та енергозалежна комп'ютерна пам'ять, що призначена для запису, зберігання та читання інформації у процесі її обробки.

Файлова база даних – база даних, що зберігається у вигляді одного або кількох пов’язаних між собою файлів із структурою, що дозволяє структурувати та розділяти дані.

Логування – процес збереження інформації щодо роботи системи в логи.

ВСТУП

Важливою частиною інформаційних систем є кешування. При обробці даних в інформаційних системах модулі кешування є високошвидкісним рівнем зберігання необхідного набору даних, який має тимчасовий характер. Доступ до даних кешу значно швидший, ніж до основного місця їх зберігання. Кешування забезпечує повторне використання раніше отриманих або обчислених даних, що забезпечує швидкодію інформаційної системи.

Швидкодія кешування залежить від методів його побудови. Методи кешування дозволяють покращити ефективність роботи інформаційних систем, а відповідно, і швидкість отримання результату за запитом до системи.

Загальним недоліком існуючих методів кешування є недостатня ступінь адаптованості до роботи інформаційних систем, в рамках яких застосовуються методи кешування. Для роботи розповсюджених методів кешування використовуються прості статистичні налаштування для виявлення необхідних для збереження даних.

Наразі для підвищення ефективності роботи методів кешування в інформаційних системах можна використовувати підбір ефективного методу кешування відповідно до характеристик інформаційної системи, а саме: кількості інформації, розміру кешу, наявності вільних ресурсів, характеру обробки даних.

Виходячи з цього, метою даної магістерської дисертації є дослідження розробленого метода адаптивного асоціативного кешування інформаційної системи з використанням асоціативних правил, які будуються на основі пошуку закономірностей у статистичних даних роботи інформаційної системи. Такий адаптивний асоціативний кеш надасть можливість покращити швидкодію інформаційної системи завдяки адаптації кешування

до правил використання системою, що можуть бути знайдені та проаналізовані з логів використання системи.

1. АНАЛІЗ ПРОБЛЕМИ

1.1. Постановка проблеми

Стандартні методи кешування не забезпечують оптимальну швидкодію інформаційних систем у випадках, пов'язаних з підвищеним навантаженням на неї. Причина цього у обмеженнях розміру кеша для його оптимальної роботи та безпосередньо у методах кешування.

Метою даного дослідження є розроблений метод адаптивного асоціативного кешування, Цей метод роботи кеша інформаційної системи при запитах на отримання даних дозволить значно збільшити швидкодію системи. Суть даного методу полягає в тому, щоб визначити типові правила взаємодії в інформаційній системі за рахунок пошуку закономірностей у відправленні різних типів запитів для оптимального використання кешу. Типові правила, або асоціативні правила збереження даних до кешу визначаються на основі адаптації статистичних даних роботи інформаційної системи, історії запитів. Визначені правила заносяться в загальний стек асоціативних правил. При наявності набору таких асоціативних правил, система буде мати можливість розміщувати результати виконання ряду запитів в кеш ще до їх виникнення. Даний метод дозволяє підвищити швидкодію роботи інформаційної системи в порівнянні з інформаційною системою з загальноприйнятими алгоритмами кешування.

Під час роботи інформаційних систем створюється багато статистичних даних про її роботу. Інформаційні системи мають потужні системи логування. Інформація зі систем логування є істотним джерелом для побудови асоціативних правил кешування.

Дослідження проводяться на розробленому програмному модулі адаптивного кешування. Наявну інформаційну систему вважатимемо такою, що містить різні функції з різними вимогами до необхідних для виклику та виконання цих функцій обчислювальних ресурсів. Кожен запит до інформаційної системи зберігається до системного логу, до якого

адаптивний модуль кешування має вільний доступ. Як зазначалось вище, методи адаптації кешу засновані на аналізі статистичних даних, які отримані під час роботи системи. Інформаційна система має декілька доступних для використання модулів для кешування – адаптивний та стандартний. Стандартний модуль кешування реалізований на основі стандартних методів кешування. На початку роботи системи використовує стандартний модуль кешування. Після того як система отримала необхідну кількість статистичних даних для запуску асоціативного адаптивного модулю, інформаційна система переходить на роботу з цим модулем кешування. Такий модуль кешування можна використовувати для покращення бистродії інформаційних систем, в яких логується історія запитів, а результат даних запитів не залежить від часу в який цей запит було отримано.

1.2. Аналіз існуючих методів кешування інформаційних систем

1.2.1. *FIFO*

Метод FIFO (First In First Out) полягає в реалізації логіки черги для оперування кешем. Відповідно дані при запиті у системи поміщуються у хвіст черги кешу. Кожний новий запит, що повертає відповідь, що ще не знаходиться у черзі кешу поміщується до хвоста кешу. Коли кеш переповнюється – досягається ліміт зберігання даних, видаляються елементи з протилежного до хвоста черги кінця [2]. Алгоритм зображений на рис. 1.

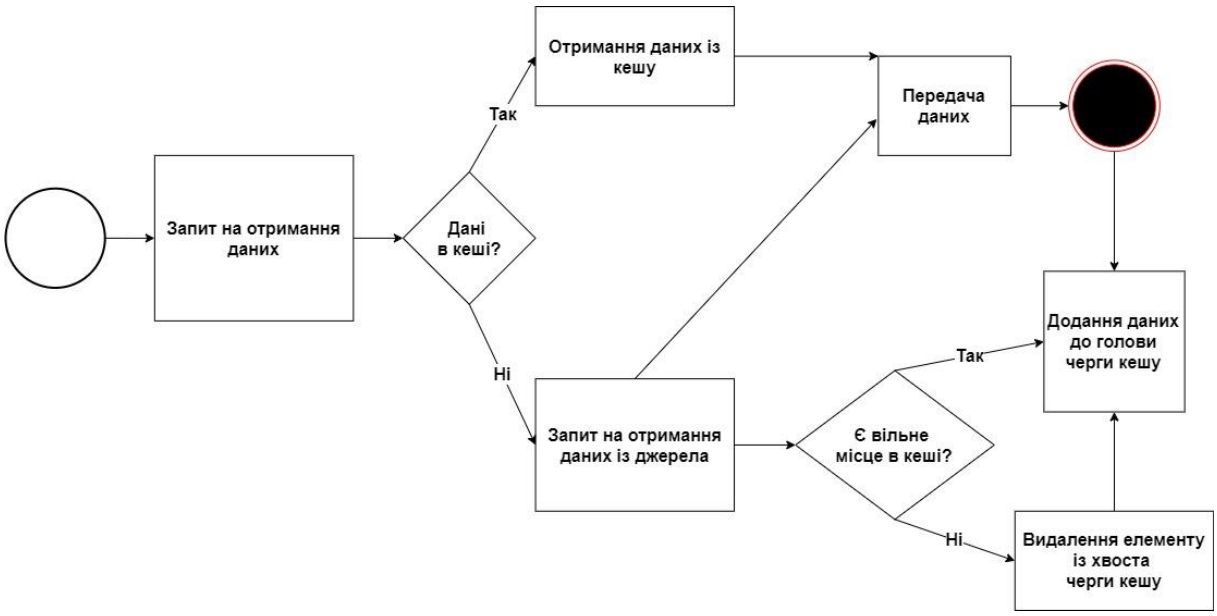


Рис. 1. Метод роботи FIFO

Перевагами такого методу є простота реалізації його у інформаційних системах. Але він і є одним з найгірших за показником ефективності роботи кешу. Дані не мають пріоритету, зберігаються послідовно та не змінюють свого положення у залежності від роботи системи та використання збережених у кеші даних. Тим не менш, такий кеш може ефективно використовуватись у частинах системи, що майже ніколи не потребують додаткового звернення, наприклад – системи авторизації користувача.

1.2.2. *LRU*

LRU (Least Recently Used) – є розповсюдженим методом кешування, що широко використовується у багатьох розподілених системах з кеш-пам'яттю. Наприклад, системи Redis та Memcached використовують саме LRU для кешування. Метод зображений на рис. 2.

Сутність даного методу кешування полягає у тому, що дані у сховищі кешованих даних відсортовані у порядку виклику. Відповідно коли дані з кешу викликаються користувачем системи, то ці дані переміщуються у початок стеку даних збережених у кеші. Нові дані також поміщуються у початок [3].

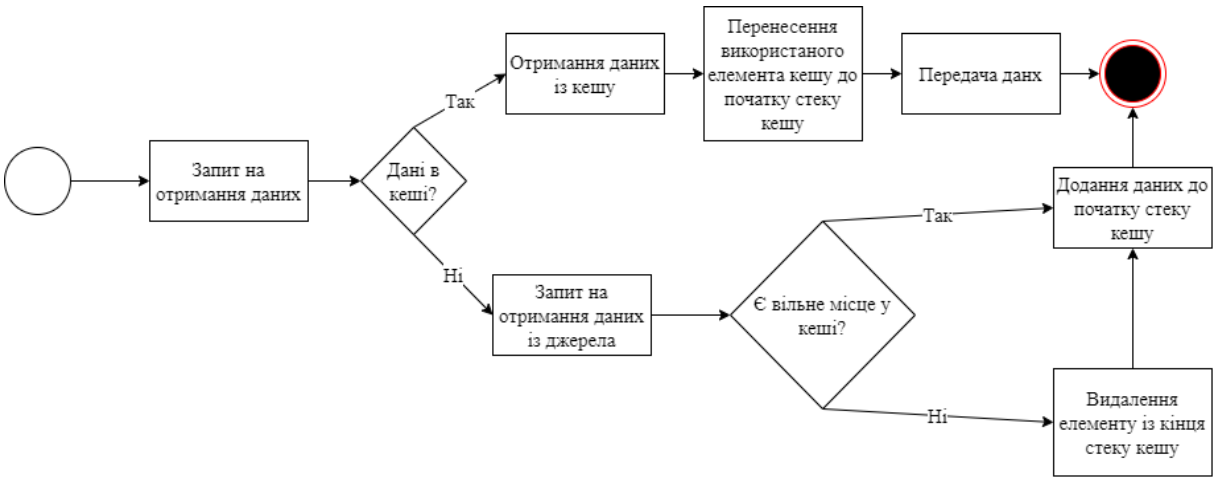


Рис. 2. Метод роботи LRU

При переповненні кешу видаляються дані з кінця кешу. Таким чином з кеша видаляються дані, що використовувались найдавніше.

Такий метод кешування набагато ефективніший за FIFO, але є вразливим до сканування системи, під час якого одноразово викликаються багато функцій, що зазвичай не використовуються.

Також існує сімейство Псевдо-LRU або PLRU кеш-методів, які покращують ефективність методу LRU, замінюючи значення, використовуючи приблизні міри віку, а не підтримуючи точний вік кожного значення в кеші.

1.2.3. *MRU*

MRU (Most Recently Used) – це альтернативний метод кешування до LRU.

Сутність даного методу кешування полягає у тому, що дані у сховищі кешованих даних відсортовані у порядку виклику. Відповідно коли дані з кешу викликаються користувачем системи, то ці дані переміщуються у початок стеку даних збережених у кеші. Такі нові дані поміщуються у початок [4].

При переповненні кешу видаляються дані з початку стеку кешування. Таким чином з кеша видаляються останні використані дані.

Для схем довільного доступу та циклічних сканувань великих обсягів даних метод MRU має кращі показники швидкодії за рахунок збереження найстаріших даних. Відповідно методи MRU найбільш ефективні у випадках коли до найстаріших елементів звертаються найчастіше.

Метод зображений на рис. 3.

Рис. 3. Метод роботи MRU

1.2.4. *LFU*

LFU (Least Frequency Used) – метод кешування, сутність якого полягає у статистичному аналізу частоти вживання даних, що зберігаються у кеші [5]. Даний метод рідко використовується з-за складності реалізації, але надає можливість уникнути недоліків звичайних LRU та FIFO методів.

Метод зображений на рис. 4.

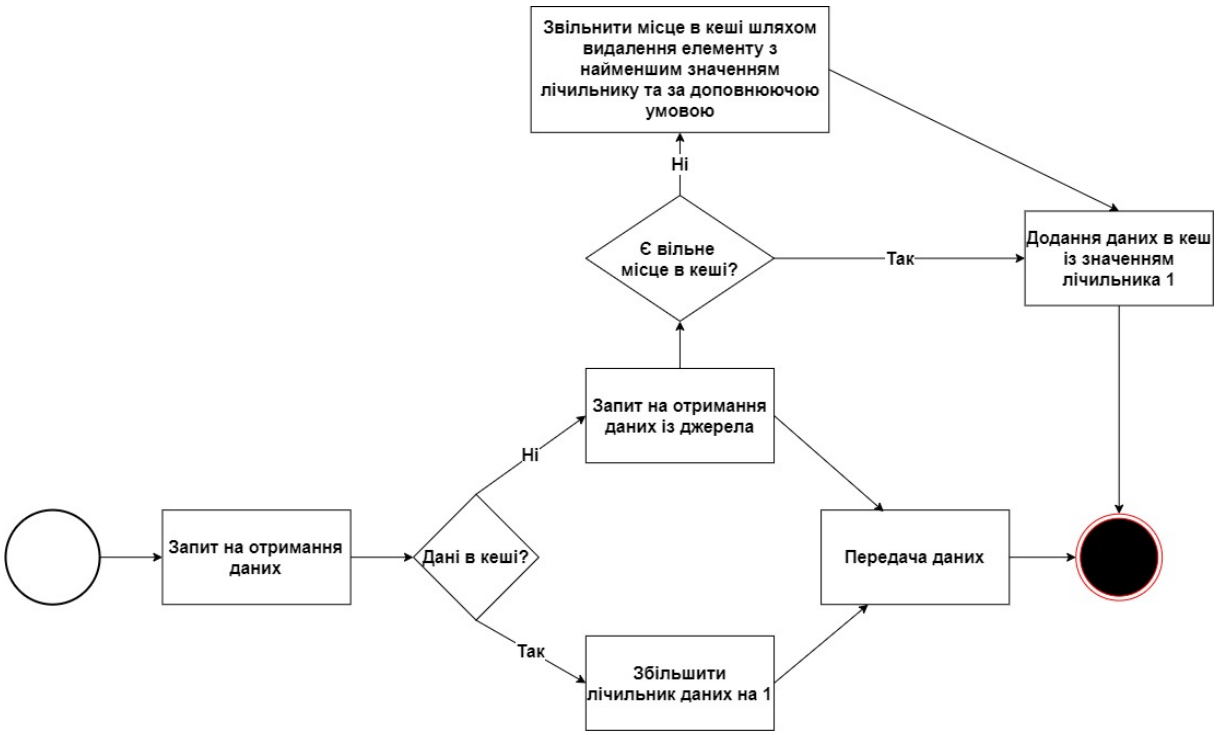


Рис. 4. Метод роботи LFU

Кожен елемент має лічильник звернень. Новий елемент вставляється в кеш зі значенням лічильника рівним 1. При попаданні в кеш лічильник знайденого елемента збільшується на 1. Якщо потрібно звільнити місце, потрібно знайти елемент з найменшим значенням лічильника. Таким чином витісняється той елемент, які запитували найрідше.

Такий метод має недоліки, що полягають у повільній адаптації кешу до зміни характеру роботи з системою. Колись частотні елементи можуть бути присутніми в кеші дуже довго, навіть якщо вони вже давно не використовуються.

1.2.5. *SN LRU*

Сегментовані LRU – Segmented LRU, або SLRU – метод кешування, що полягає у створенні декількох (двох або більше) сегментів, за своїм методом роботи ідентичних до LRU кешу. Метод графічно представлений на рис. 4.

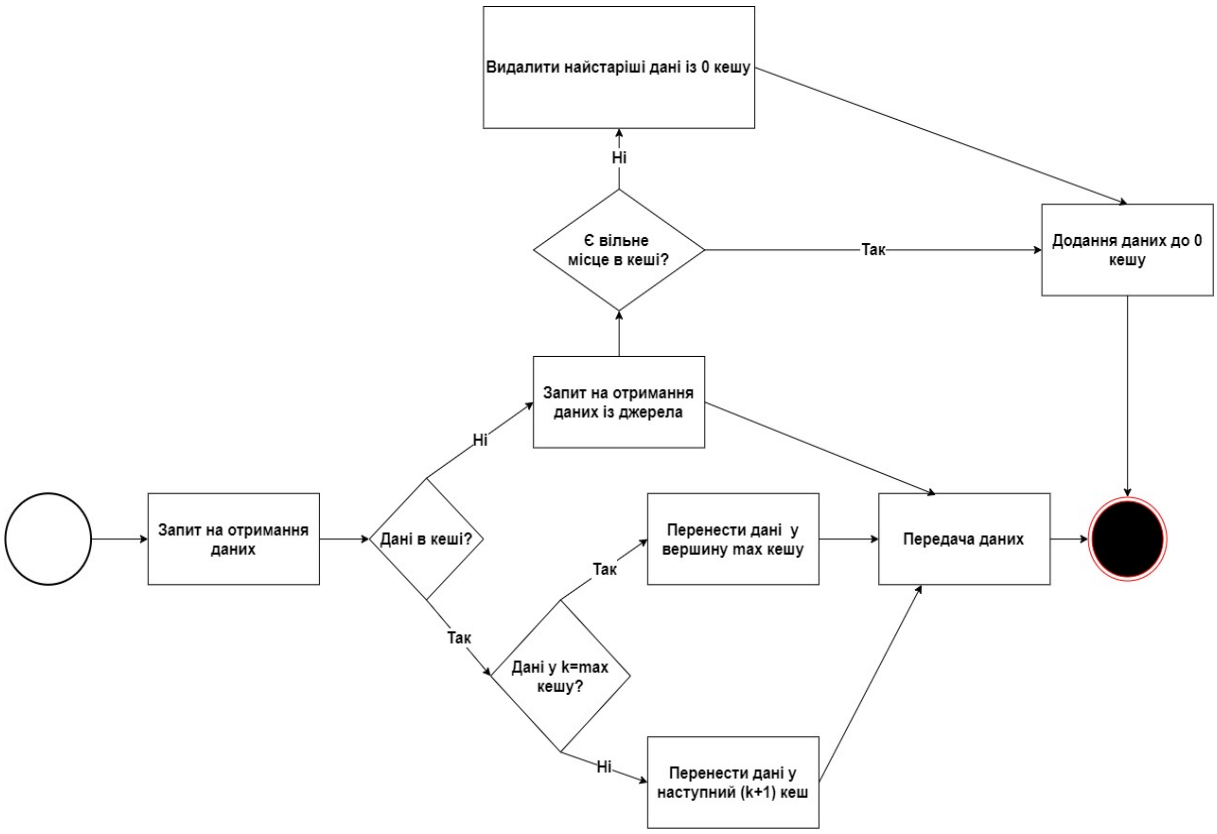


Рис. 4. Метод роботи Segmented LRU

Дані, що попадають у кеш, попадають до першого сегменту. Якщо дані потім не використовуються, дані з першого сегменту з часом видаляються для збереження нових даних. Якщо дані, збережені у кеші, використовуються, то такі дані переміщуються у початок наступного сегменту. Таким чином утворюється структура, у якій є найбільш вживані елементи – у сегментах з великим номером, і дані що використовуються рідко – у сегментах з низькими номерами [6]. Приклад сегментації зображений на рис. 5.

Такий метод кешування є стійкою до сканування системи. Також такий метод можна назвати певного роду адаптацією до роботи інформаційноїх системи. Але в даному методі відсутня пріоритезація даних що зберігаються, і також наявне старіння інформації, що зберігається у високих сегментах такого кешу, так як видаляються дані з високих сегментів досить повільно.

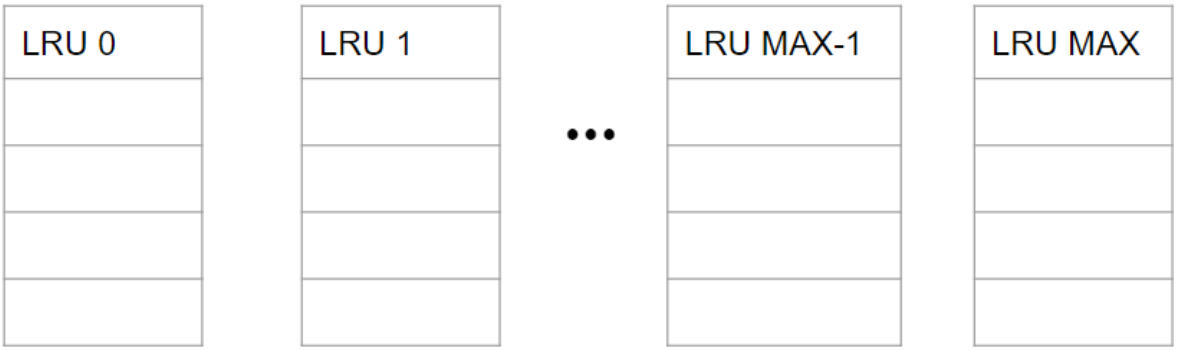


Рис. 5. Сегменти SN LRU кешу

1.2.6. *ARC*

Кеш адаптивного заміщення – Adaptive Replacement Cache – це метод кешування з заміною елементів з кращою продуктивністю, ніж LRU. Це досягається шляхом відстеження як часто використовуваних, так і нещодавно використовуваних елементів, а також нещодавньої історії виселення обох [7]. Метод був розроблений в Дослідницькому центрі IBM Almaden. У 2006 році IBM отримав патент на політику адаптивного кешування заміщення.

ARC вдосконалює основну стратегію LRU, розділяючи каталог кеш-пам'яті на два списки, T1 і T2, для нещодавно записаних записів. У свою чергу, кожен з них поширюється списком привидів (B1 або B2), який додається внизу двох списків. Ці списки привидів виконують роль показників, відстежуючи історію нещодавно виселених записів кеш-пам’яті, а метод використовує примарні звернення для адаптації до нещодавніх змін у використанні ресурсів. Зверніть увагу, що списки привидів містять лише метадані (ключі для записів), а не самі дані ресурсу, тобто, коли запис виселяється у список привидів, його дані відкидаються. Комбінований каталог кешу складається з чотирьох списків LRU:

- T1, для останніх записів кешу.
- T2, для частих записів, посилання на щонайменше двічі.
- B1, записи привидів, нещодавно виселені з кешу T1, але все ще відстежуються.

● B2, подібні записи привидів, але виселені з T2.

T1 і B1 разом позначаються як L1, об'єднана історія останніх окремих посилань. Подібним чином L2 є комбінацією T2 і B2.

Виконаємо порівняльний аналіз найбільш поширених методів кешування. Оцінка методів кешування за 10 бальною шкалою наводиться у табл. 1.

Таблиця 1

Оцінка методів кешування

Метод кешування	Ефективність при випадкових викликах елементів інтерфейсу	Ефективність при наявності послідовностей при викликах елементів інтерфейсу	Простота реалізації	Додаткове навантаження системи
FIFO	3	1	10	1
LRU	3	5	8	4
LFU	3	5	8	5
MRU	3	4	8	5
SN LRU	3	6	5	7
ARC	3	8	4	8

Аналіз статистики використання системи потребує додаткових апаратних ресурсів, а також такі методи складніше у реалізації. Ефективність при випадковому використанні інтерфейсу інформаційної системи однакова так, як ніяка статистична інформація не надає переваги для виявлення найбільш вигідних для збереження даних. З таблиці 1 видно, що чим ефективніше методи кешування пристосовуються до патернів та правил використання системи, тим складніше реалізувати дані методи.

1.3. Актуальність задачі

Кешування може бути ефективним інструментом роботи інформаційної системи, а саме в частині: зниження навантаження, підвищення продуктивності додатків, підвищення пропускної спроможності операцій введення-виведення, набагато більш високу швидкість виконання запитів. Кешування може покращити бистродію як різноманітних систем, так і складних математичних алгоритмів, що потребують значних ресурсів для виконання [8].

Основною проблемою розглянутих вище методів кешування є відсутність адаптації роботи кешу до роботи системи. Стандартне кешування основане на простих статистичних концепціях має недоліки, пов'язані з відсутністю пріоритету кешування, слабкою стійкістю до старіння інформації та орієнтованість на частоту влучень, а не ефективність роботи кешу. Не використовується аналіз патернів запитів для адаптації роботи кешу.

Також загальним недоліком для методів що використовують статистичну інформацію для адаптації роботи кешу, як наприклад *ARC* та *LRU,* можна визначити те, що аналіз проводиться постійно. Для зменшення необхідних для адаптації ресурсів можна проводити глибокий аналіз лише за необхідністю, наприклад – коли становиться меньше влучень у кеш, або через деякі проміжки часу.

Недоліки існуючих методів кешування потребують покращення і розроблення нових методів кешування, зокрема – адаптивного асоціативного кешу.

Виходячи з цього, є актуальною робота присвячена дослідженню розробленого метода адаптивного кешування інформаційної системи з використанням асоціативних правил, які будуються на основі статистичних даних роботи інформаційної системи. Такий адаптивний кеш надасть можливість покращити швидкодію інформаційної системи завдяки

адаптації методу кешування до правил використання системою, що можуть бути знайдені та проаналізовані з логів використання системи.

Таким чином, предметом даної роботи є розробка та дослідження методу адаптивного асоціативного кешування, побудованого на визначенні типових правил взаємодії в інформаційній системі за рахунок пошуку закономірностей у відправленні різних типів запитів для оптимального використання кешу.

1.4. Огляд існуючого програмного забезпечення для кешування інформаційних систем

1.4.1. *Memcashed*

Memcashed – програмне забезпечення, що реалізує сервіс кешування даних в оперативній пам'яті на основі хеш-таблиці [9].

За допомогою клієнтської бібліотеки (для C / C ++, Ruby, Perl, PHP, Python, Java, .Net і ін.) дозволяє кешувати дані в оперативній пам'яті безлічі доступних серверів. Розподіл даних по серверам реалізується шляхом сегментування даних за значенням хешу ключа за аналогією з сокетами хеш-таблиці. Клієнтська бібліотека, використовуючи ключ даних, обчислює хеш і використовує його для вибору відповідного сервера. Ситуація збою сервера трактується як cache miss, що дозволяє підвищувати відмовостійкість комплексу за рахунок нарощування кількості memcached серверів та можливості виробляти їх заміну.

В API memcached є тільки базові функції: вибір сервера, установка і розрив з'єднання, додавання, видалення, оновлення і отримання об'єкта, а також Compare-and-swap. Для кожного об'єкта встановлюється час життя, від 1 секунди до нескінченності. При вичерпанні пам'яті старіші об'єкти автоматично видаляються. Для PHP також є вже готові бібліотеки PECL для роботи з memcached, які дають додаткову функціональність.

1.4.2. *Redis*

Redis – резидентна система управління базами даних класу NoSQL з відкритим вихідним кодом, що працює зі структурами даних типу «ключ-значення» [10]. Використовується як для баз даних, так і для реалізації кеша, брокерів повідомлень.

Орієнтована на досягнення максимальної продуктивності на атомарних операціях (заявляється про приблизно 100 тис. SET- і GET-запитів в секунду на Linux-сервері початкового рівня). Написана на Сі, інтерфейси доступу, створені для більшості основних мов програмування.

У період 2010-2013 років розробка системи спонсорувалася компанією VMware, з травня 2013 року, після реорганізацій в федерації EMC-VMware, проєкт переданий в Pivotal. З червня 2015 року основний спонсор проєкту – компанія Redis Labs, спеціально заснована для комерціалізації Redis, в неї ж перейшов основний розробник продукту – Сальваторе Санфіліппо.

1.4.3. *Система кешування Django*

Django – вільний фреймворк для веб-додатків на мові Python, що використовує шаблон проєктування MVC.

Інтерес викликає система кешування, що реалізована усередині [11].

Django має можливості використання широкої варіації кешування. Це і веб-кешування, і клієнт-кешування, серверне, баз даних. Залежить від налаштувань під час підключення.

В зазначеному вище програмному забезпеченні можна використати адаптивний асоціативний кеш. Для Redis та Memcashed можливе розгортання даного модулю кешування на кожному з сегментів, а в Django вже існує функціонал для підключення власних модулів кешування.

1.5. Висновки до розділу

У даному розділі було розглянуто проблему ефективності кешування для інформаційних систем, що мають багато користувачів та потужний потік запитів до системи. Розглянуто необхідність створення методу асоціативного адаптивного кешування даних інформаційної системи, що дозволить підлаштовуватись під окремих користувачів системи та покращувати швидкодію.

Розглянуте існуюче програмне забезпечення для забезпечення кешування інформаційних систем.

Було розглянуто існуючі методи кешування даних: на основі черги, на основі статистичних показників, на основі частоти використання, їх модифікації. Проаналізовано їх переваги та недоліки. До основних недоліків відносяться:

- Відсутність пріоритету кешування.
- Не чутливість до старіння даних.
- Відсутність аналізу інформації логування роботи інформаційної системи для адаптації методів кешування.

Для подолання наведених вище недоліків необхідно створити метод кешування даних у інформаційних системах, який буде мати змогу визначати типові правила взаємодії в інформаційній системі за рахунок пошуку закономірностей у відправленні різних типів запитів. Типові правила, або асоціативні правила збереження даних до кешу визначатимуться на основі аналізу статистичних даних роботи інформаційної системи, історії запитів.

2. МЕТОД ПОБУДОВИ МОДЕЛІ АДАПТИВНОГО АСОЦІАТИВНОГО КЕШУ В ІНФОРМАЦІЙНИХ СИСТЕМАХ

Враховуючи недоліки існуючих методів, необхідно створити модель адаптивного асоціативного кешу, яка враховує історію запитів до інформаційної системи і адаптується у відповідності до взаємозв'язків між цими запитами, їх параметрами.

Інформаційна система, що у повній мірі може використовувати створену модель повинна відповідати певним вимогам. Під інформаційною системою у роботі розуміється система з наступними особливостями:

- Система може отримувати водночас багату кількість запитів.
- Інтерфейс системи складається з великої кількості функцій з параметрами та без них.
- Інформація, що надається системою у відповідь на виклик певних функцій не втрачає своєї актуальності певний час.
- Кожен виклик функцій логується разом з параметрами виклику.
- Кількість інформації в системі достатньо велика, щоб була потреба у ефективній системі кешування.
- Розмір буферу кешу менший за потенційний об'єм інформації що можна отримати в системі.

Система такого виду була обрана як еталонний варіант системи для впровадження адаптивного асоціативного кешу.

2.1. Опис адаптивної асоціативної моделі кешування інформаційної системі

Для підвищення ефективності роботи інформаційної до систем було створено метод адаптивної асоціативного кешування.

Інформаційна система, яка використовує модуль адаптивного асоціативного кешування. Графічне зображення такої інформаційної системи наводиться на рис. 6. має відмінності від загальної моделі

інформаційної системи. Інформаційна система, яка використовує модуль адаптивного асоціативного кешування, складається з наступних компонентів:

- Джерело даних.
- Інтерфейс інформаційної системи.
- Менеджер даних.
- Адаптивний кеш.
- Лог запитів.

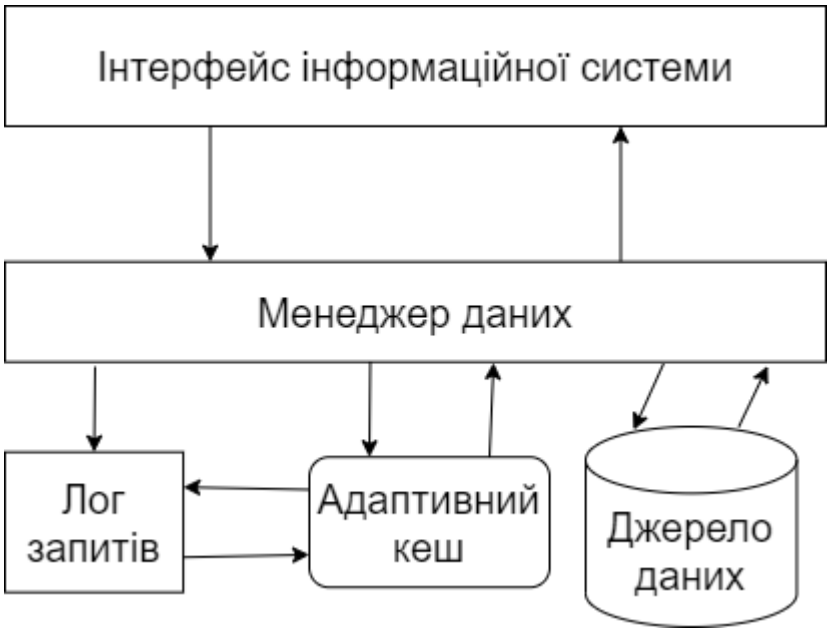


Рис. 6. Модель системи з адаптивним асоціативним кешем

Джерелом даних може бути база даних або інше структуроване сховище інформації.

Інтерфейс системи це набір запитів, який може бути використаний під час поточної роботи інформаційної системи. До менеджера даних надходить набір запитів, що забезпечує їх виконання в тому порядку, який визначається конфігурацією менеджера. Менеджер даних є єдиною частиною інформаційної системи, що розділяється інтерфейсом та управляє джерелом даних, адаптивним кешем, логом запитів. Менеджер даних вбудовує отримані запити в чергу виконання відповідно порядку

надходження. Менеджер даних вибирає звідки взяти дані, в першу чергу звертаючись до кешу, і якщо даних в кеші не виявилося, то слідує звернення до джерела даних. Кожне звернення до менеджера з боку інтерфейсу фіксується і поповнює лог запитів.

Адаптивний кеш – кеш, для адаптації до умов використання системи використовує лог запитів.

Логом запитів є історія запитів користувачів і параметри таких запитів.

Модель системи з адаптивним кешем відрізняється від звичайної інформаційної системи наявністю нових модулів – менеджеру даних та логом запитів.

Завдяки наявності лога запитів система має можливість підлаштуватись до правил використання інтерфейсу. У ролі логу даних може слугувати файлова або орієнтована на бази даних система логування, яка буде логувати виклики та запити користувачів, параметри цих викликів.

Для ефективної взаємодії, інтеграції з адаптивною моделлю кешування логування, зберігання історії запитів повинне бути у вигляді, що може бути просто відсортований та отриманий з боку менеджера даних. Якщо логування буде відбуватись на рівні файлів, то розбір ресурсів файлів може бути занадто ресурсоємним для ефективної роботи моделі. Окрім того інформаційна система повинна мати просту для аналізу систему логування.

Менеджер даних включає в себе модуль адаптації, що керує даними у кеші, а також може за необхідності відправляти додаткові запити до джерела даних.

2.2. Особливості реалізації ассоціативної моделі кешування інформаційної системі

Для адаптації кеша до характеру використання системи використовуються асоціативні правила та їх побудова [12]. Загальний алгоритм побудови наведений на рис. 7. Робота з асоціативними правилами

дозволяє знайти закономірності у взаємодіях користувачів над системою. Також побудова асоціативних правил може бути набагато швидшою за використання машинного навчання на тому самому наборі даних.

Асоціативні правила – встановлення типових взаємозв'язків між елементами всередині датасета. Ці правила лягають в основу прогнозування майбутніх дій системи. Наприклад: "Виклик функції x, з великою часткою ймовірності призведе до виклику функції y".

Розглянемо знаходження асоціативних правил в послідовності запитів користувача до джерела даних. Є певний набір запитів, доступний користувачеві. Він наведений у табл. 2.

Таблиця 2

Набір запитів, доступний для виклику користувачем

ІНДЕКС ЗАПИТУ	ОПИС ЗАПИТУ
0	query0
1	query1
2	query2
3	query3
4	query4
5	query5
6	query6
7	query7
8	query8
9	query9

Необхідно знайти асоціативні правила для даного набору запитів.

Введемо позначення:

G: i_1-i_2-i_3 -...- i_n – опис транзакції, де G – номер транзакції, а i_k – індекс типу запиту.

$D(i)$ – безліч транзакцій, які включають в себе запит з індексом i.

До бази було вироблено 5 транзакцій з такими наборами запитів:

1: 0-1-2-1-5

2: 9-8-7-5

3: 3-1-2

4: 4-5-7

5: 1-2-3-6

Тоді:

$D(1)$ = 1, 3, 5

$D(2)$ = 1,3,5

$D(3)$ = 3, 5

$D(4)$ = 4

$D(5)$ = 1,2,4

$D(6)$ = 5

$D(7)$ = 2, 4

$D(8)$ = 2

$D(9)$ = 2

Конвертуємо набір транзакцій до бінарного виду:

1: 0-1-2-1-5

2: 9-8-7-5

3: 3-1-2

4: 4-5-7

5: 1-2-3-6

Отримуємо представлення входження запитів до транзакцій. Воно наведено у табл. 3.

Таблиця 3

Представлення входження запитів до транзакцій

Номер транзакції	Query 1	Query 2	Query 3	Query 4	Query 5	Query 6	Query 7	Query8	Query9
1	1	1	0	0	1	0	0	0	0
2	0	0	0	0	1	0	1	1	1
3	1	1	1	0	0	0	0	0	0
4	0	0	0	1	1	0	1	0	0
5	1	1	1	0	0	1	0	0	0

В теорії про асоціативні правила існують основні поняття для опису і оцінки правил:

- Support.
- Confidence.
- Lift.
- Conviction.

Алгоритми пошуку займаються пошуком правил XY (Сталося X – трапиться і Y). Відбір правил відбувається за допомогою оцінки чисельних показників support, confidence, рідше – Lift і Conviction, які не повинні бути нижче заздалегідь встановленого порога.

Знаходження Support:

$$supp(X) = \frac{|D(X)|}{|T|} \quad (1)$$

X-набір Query, який досліджується на наявність правила. | D (X) | – кількість транзакцій, в яких є даний набір queries, а | T | – загальна кількість транзакцій.

Отже, Support – це частотний показник досліджуваного набору запитів в даному наборі транзакцій.

Дослідження Confidence:

Confidence – це достовірність досліджуваного правила, яка вираховується за формулою:

$$conf(X) = \frac{sup(X)}{sup(q_i, q_i \in X)} \quad (2)$$

Confidence правила "при виклику запиту q_i, викликаються інші запити з X"

Знаходження Lift:

$$lift(qi \cup qj) = \frac{supp(q_i \cup q_j)}{supp(q_i) * supp(q_j)} \quad (3)$$

Lift в асоціативних правилах служить величиною вимірювання залежності одного дії від іншого, в нашому випадку – залежність q_i від q_j. Обчислюється відповідно розподілом значення support для досліджуваного набору на значення support включених в даний набір елементів.

Якщо значення lift = 1, то дії є незалежними, якщо менше 1 – то правила негативно впливають на залежності.

Якщо значення lift > 1, то величина, на яку lift більше 1 показує силу данного правила.

Знаходження Conviction:

Conviction – це значення, що визначає частоту "відсутність спрацьовування" правила, що розглядається.

$$conv(q_i \cup q_j) = \frac{1 - supp(q_j)}{1 - conf(q_i \cup q_j)} \quad (4)$$

Класичні алгоритми, такі як брутфорс-алгоритм, Apriori-алгоритм, ECLAT-алгоритм, FP-growth використовують перераховані поняття дозволяють для знаходження асоціативних правил в наборах даних.

Розглянемо і оцінимо правило "Якщо відбувається запит q_1 то відбувається і запит q_3" за допомогою перерахованих понять.

$$\sup(q_1 \cup q_3) = \frac{2}{5} \quad (5)$$

Confidence для правила, що при виклику q_1 буде викликано q_3:

$$conf(q_1 \cup q_3) = \frac{\frac{2}{5}}{\frac{3}{5}} = \frac{2}{3} \quad (6)$$

$$lift(q_1 \cup q_3) = \frac{\frac{2}{5}}{\left(\frac{3}{5} * \frac{2}{5}\right)} = \frac{10}{6} \quad (7)$$

$$conv(q_1 \cup q_3) = \frac{1 - supp(q_1)}{1 - conf(q_1 \cup q_3)} = \frac{6}{5} \quad (8)$$

У адаптивному ассоциативному кешу для механізму адаптації використовується аналіз журналу транзакцій. Результатом такого аналізу є набір асоціативних правил, який може використовувати менеджер даних для підтримки механізму адаптації.

Параметрами транзакцій виступають виклики функцій. У роботі однакові функції, що викликаються з різними параметрами під час будування асоціативних правил рахуються різними параметрами.

У інформаційній системі немає можливості однозначно відокремити цикли роботи, так як вона працює безперервно, то постає проблема розбиття потоку запитів на транзакції для аналізу. Тому використовується метод розбиття запитів на транзакції за часом зі зрушеннями в часі.

Зрушення під час вибору транзакцій використовуються для уникнення проблем з обранням “невдалого розподілу часу”. Якщо системою користується інша автоматизована система, то її робота з інформаційною системою може бути чітко прив’язана до часу. Відповідно для того, щоб уникнути ситуації, коли транзакції обрані невдало для виявлення взаємозв’язків у викликах функцій автоматизованою системою, проміжки часу які обираються для відокремлення однієї транзакції вибираються випадковим чином в межах допустимих значень.

Межі допустимих значень часових кордонів транзакцій визначаються відповідно до завантаження системи запитами. Відповідно, при більшої кількості запитів – межі можуть бути вужчими.

Рис. 7. Загальний алгоритм побудови асоціативних правил

2.3. Визначення асоціативних правил

Для пошуку асоціативних правил був обраний апріорний алгоритм пошуку асоціативних правил [13, 14].

Даний алгоритм є одним з найбільш відомих алгоритмів для пошуку асоціативних правил. Даний алгоритм використовує властивість анти-монотонності, і завдяки цьому має можливість обробки великої кількості транзакцій та знаходження асоціативних правил за прийнятний час.

Для того щоб можна було використовувати алгоритм, проводиться обробка та аналіз транзакцій.

Транзакції заносяться до таблиці вигляду табл. 4.

Таблиця 4

Представлення транзакцій системи у таблиці

TID	A	B	C	D	E	...
1	1	1	0	0	1	0
2	0	0	0	0	1	0
3	1	1	1	0	0	0
4	0	0	0	1	1	0
5	1	1	1	0	0	1

TID – номера транзакцій, A, B, C, D, E, ... – інтерфейс інформаційної системи. Відповідно на перетині номера транзакцій та елементу інтерфейсу знаходиться значення використання або ігнорування елементу інтерфейсу у рамках однієї транзакції.

Знаходження найбільш вживаних взаємозв'язків елементів інтерфейсів досить ресурсоємна задача, що потребує значної кількості часу. Для прискорення даного процесу апріорний алгоритм побудови асоціативних правил використовує властивість змінної підтримки. Підтримка будь-якого набору елементів не перевищує мінімальної підтримки будь-якого з підмножин даного набору. Дану властивість називають властивістю анти-монотонності.

Візьмемо мінімальне прийнятне значення підтримки рівне 50%. Відповідно маємо наступне дерево знаходження асоціативних правил за апріорним алгоритмом відповідно до таблиці 2 на рис. 8.

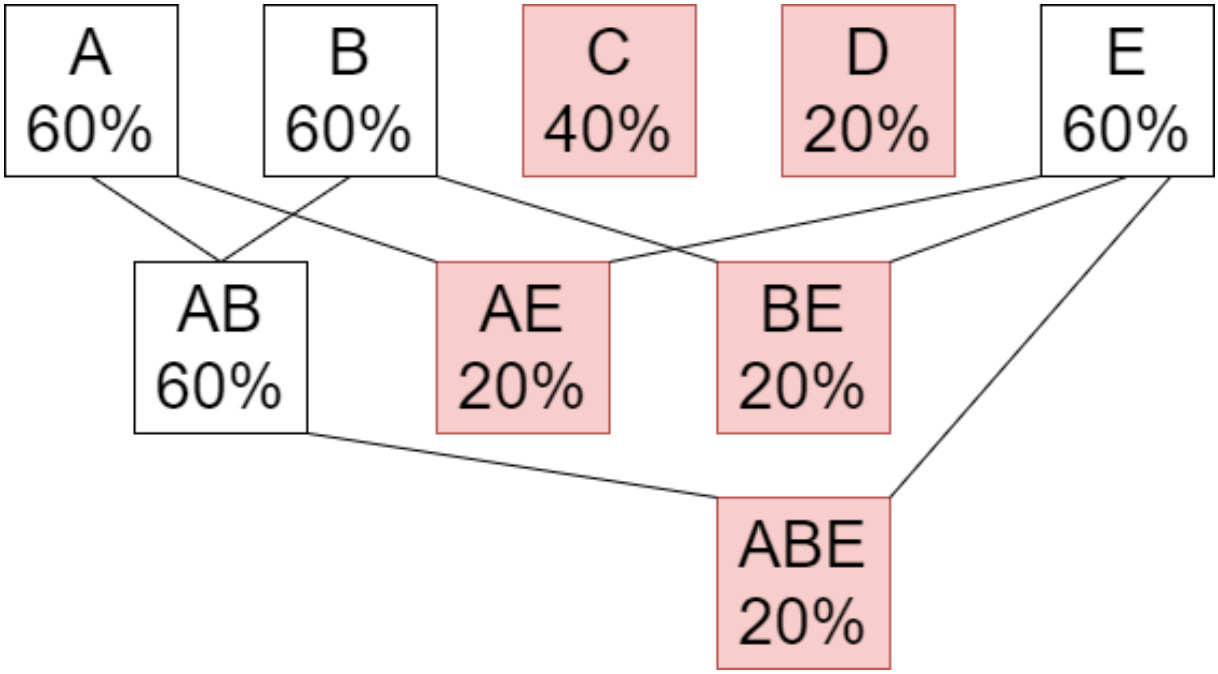


Рис. 8. Дерево побудови асоціативних правил за апріорним алгоритмом

Метод побудови асоціативних правил за апріорним алгоритмом відсікає необхідність у аналізі елементів та їх груп, що не задовольняють граничному значенню Support, відповідно швидкість аналізу значно підвищується.

Відповідно до рисунку 8, було знайдено асоціативне правило – AB, та елементи що задовольняють граничному значенню підтримки – A, B, E.

Асоціативні правила для роботи адаптивного асоціативного кешу будуються на основі логу запитів, наявного у системі.

Також для покращення якості адаптації системи, можна додати окремі групи асоціативних правил:

- Правила необхідності на додавання.
- Правила дозволу на видалення.
- Правила заборони на видалення.

Кожен з наборів правил має свою функцію при роботі адаптивного асоціативного кешу.

2.4. Адаптація кешу з використанням асоціативних правил. Додавання та видалення даних

Кеш може дозволити додавання набору даних, якщо в нього є пусте місце, так набір даних становиться “кандидатом на додавання”. Необхідність додавання даних до кешу вираховується за рахунок асоціативних правил необхідності на додавання. Ці правила знаходять зв’язок між кандидатом, уже збереженими даними, та імовірністю подальшого використання кандидата під час роботи системи. Якщо даний зв’язок має достатні для збереження значення, то дані зберігаються до адаптивного асоціативного кешу.

Коли кеш вважається “заповненим” то виникає необхідність у видаленні деяких елементів з кешу, які будуть використані з найменшою вірогідністю. В такому разі аналізується наявний набір асоціативних правил для отримання дозволу на видалення певних даних. Це досягається шляхом знаходження елементів кешу із найменшими асоціативними зв’язками з іншими елементами кешу.

При видаленні елементів можуть враховуватись додаткові правила заборони на видалення. Вони не дають видалити елементи, що можуть бути пов’язані з уже збереженими елементами усередині кешу. Такі правила будуються на основі аналізу взаємозв'язку кандидата на видалення, поточних параметрів системи, даних що потребують збереження. Сутність цих правил полягає у забороні на видалення таких елементів з кешу, без яких не можуть бути використані збережені елементи, що не є кандидатами на видалення.

Правила будуються на основі єдиного логу запитів, але кожен з наборів правил може будуватись з різними параметрами.

Параметрами при побудові або перебудові асоціативних правил можуть бути:

- Частота перебудови. Як часто система вважає набір правил “застарілими” та потребує їх перебудування.
- Кількість транзакцій, що враховується. Транзакції зберігаються за великий час, але кожен з наборів правил може вибирати період, транзакції якого потрібно враховувати.
- Підтримка правила. Параметр, що відповідає ймовірнісний оцінці роботи правила у окремо взятій транзакції. Дані параметри залежать як від типу цільового набору правил та від характеру системи що розглядається.

Отже, у інформаційній системі при роботі з адаптивним асоціативним кешем виділено кроки:

1. Отримання запиту менеджером даних з боку інтерфейсу.
2. Збереження запиту до логу.
3. Формування запиту до адаптивного асоціативного кешу. Якщо адаптивний кеш повернув дані, переходимо до кроку 6.
4. Формування запиту до джерела інформації даних.
5. Отримання даних. Дані також надсилаються до кешу. Кеш вирішує, чи зберігати йому отримані дані, та видаляє за необхідністю застарілі дані з свого сховища інформації.
6. Отримання даних інтерфейсом системи.

Графічно із даними кроками можна ознайомитись на рис. 9.

Відповідно задачею використання асоціативних правил під час роботи з кешем є мінімізація кількості запитів до джерела даних, що розвантажує роботу джерела інформації.

Даний підхід для роботи кешу дозволяє точно знаходити застарілі елементи, що потребують видалення, а також вирішувати які елементи будуть використовуватись з більшою ймовірністю. За допомогою асоціативних правил кеш може прогнозувати необхідність збереження

даних до кешу, а саме – будувати відповідність між параметрами роботи системи та даними що проходять через систему, а також між параметрами запитів на дані.

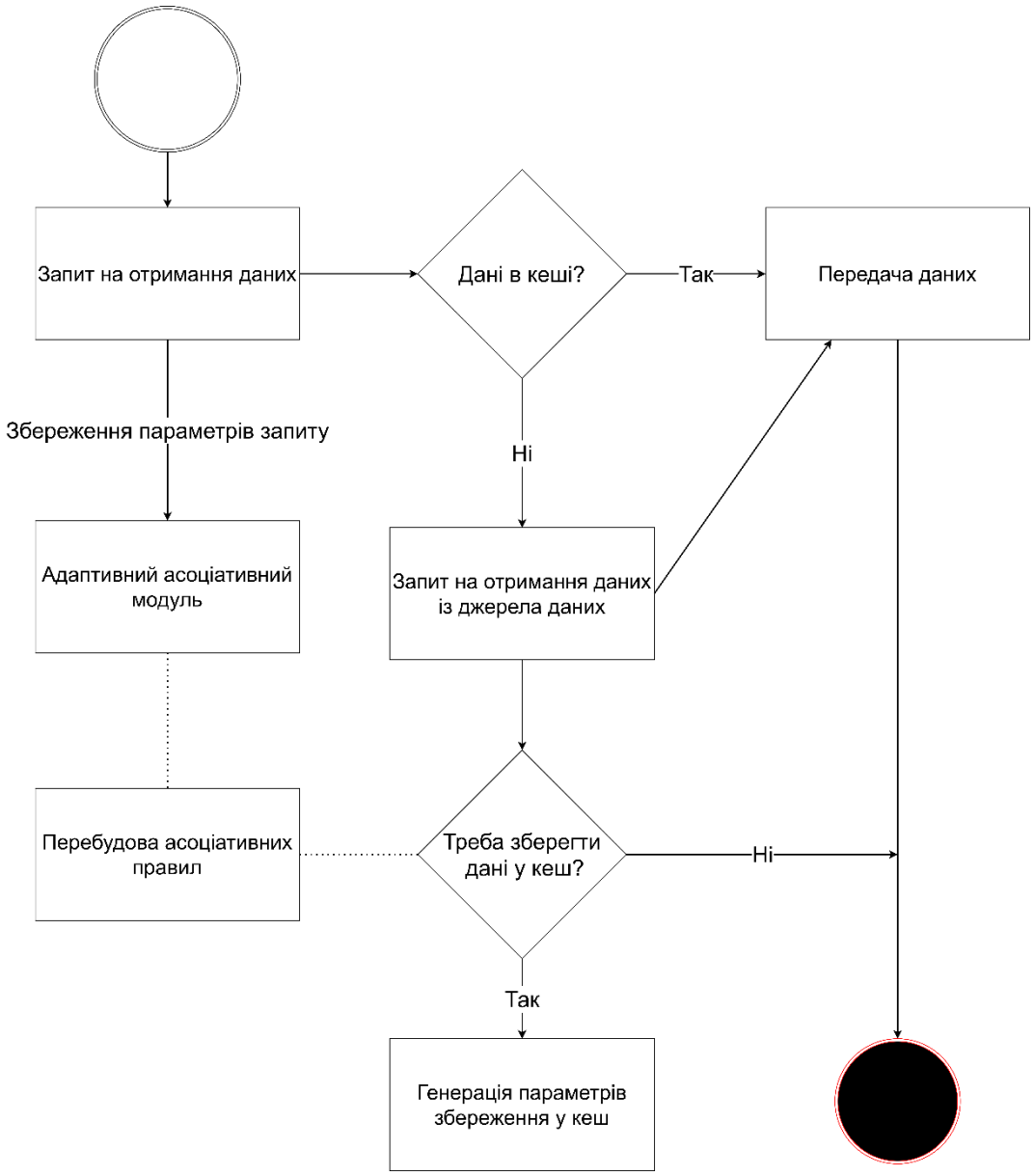


Рис. 9. Метод отримання даних з адаптивного асоціативного кешу при отриманні запиту

2.5. Адаптація кешу з використанням асоціативних правил. Превентивне додавання даних до кешу

Менеджер даних має можливість посилати при необхідності додаткові запити для отримання даних до джерела даних. Така можливість менеджеру

даних дозволяє з використанням асоціативних правил підключити превентивне додавання даних до кешу.

Коли у системі викликається функція X, при чому наявний взаємозв'язок – асоціативне правила – між функцією X та Y, можна зробити припущення, що у тій самій транзакції – проміжку часу – буде викликано функцію Y.

Таким чином менеджер даних, без очикування безпосереднього запиту Y від користувача може викликати функцію Y. Результат такої операції буде покладений до кешу, у якому він буде очікувати безпосереднього запиту від користувача.

Таким чином, швидкість системи може бути покращена безпосередньо на той самий час, що відокремлює виклик функції Y від безпосереднього запиту на отримання результату виконання функції Y.

2.6. Аналіз алгоритму роботи інформаційної системи з адаптивним асоціативним кешем

Алгоритм роботи інформаційної системи складається з взаємодії раніше розглянутих сутностей: інтерфейсу системи, менеджеру даних, адаптивного асоціативного кешу, джерела даних, логу запитів – та взаємодії між ними.

На початку роботи з даними стоїть інтерфейс інформаційної системи. Від надсилає запит на отримання даних до менеджеру даних.

Менеджер даних зобов'язаний зберегти даний запит до логу запитів та відправити трансформований запит до адаптивного кешу. Адаптивний кеш у свою чергу перевіряє наявність усередині відповіді, за її наявності – віддає дані безпосередньо користувачу.

Якщо необхідні дані усередині відсутні, запит на їх отримання перенаправляється до джерела даних. Джерело даних обробляє дані, віддає необхідну статистичну та іншу інформацію асоціативному кешу, віддає дані інтерфейсу системи.

При певних модифікаціях віддача даних інтерфейсу може також проводитися через менеджер даних. Такий метод роботи дозволяє відокремити роботу окремих модулів. Лог запитів поповнюється окремо від роботи адаптивного модуля. Адаптивний модуль відокремлений від менеджера даних та може мати декілька реалізацій у залежності від загальних особливостей системи.

Графічне відображення архітектури такої системи зображене на рис.10.

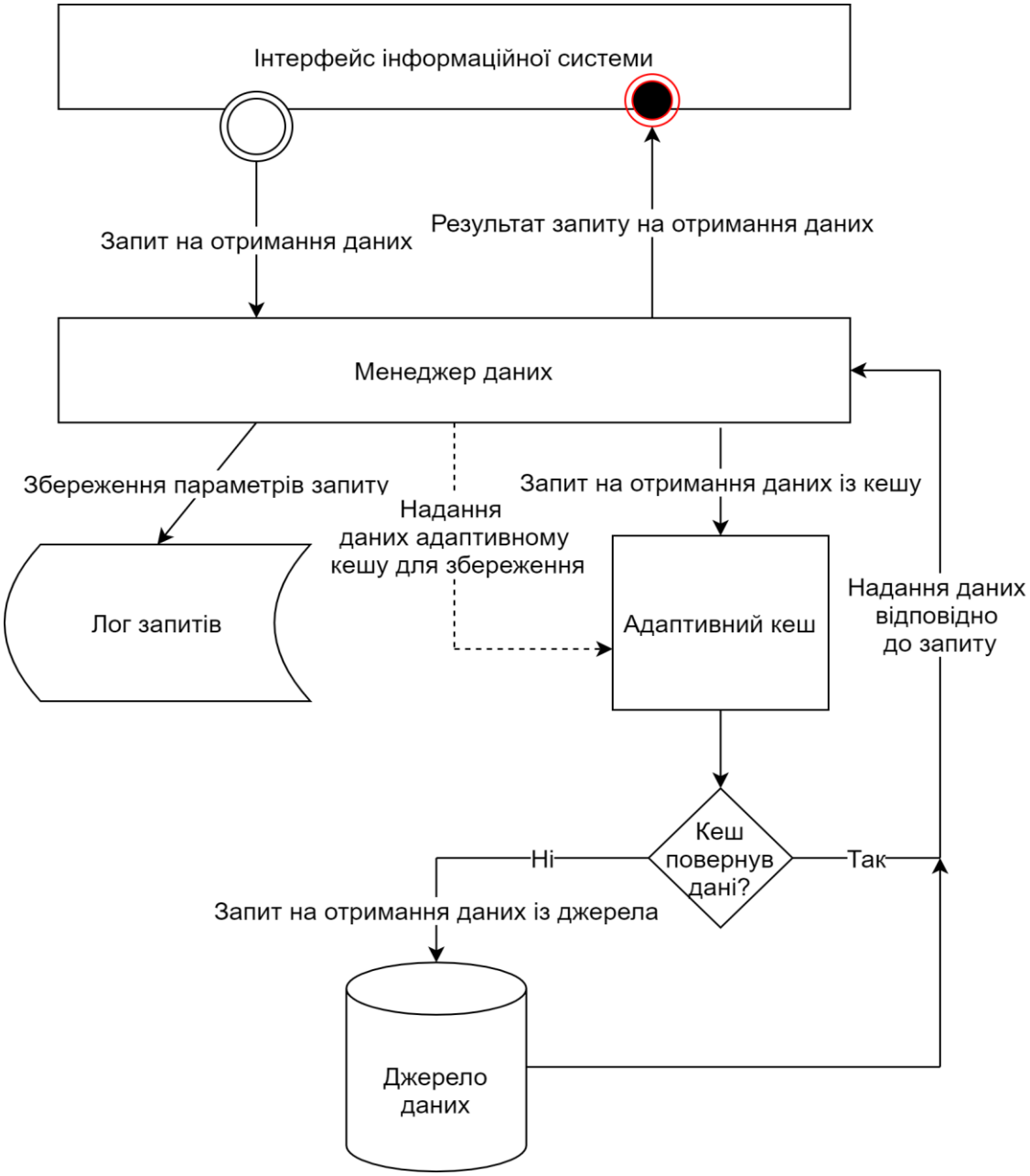


Рис. 10. Архітектура інформаційної системи з адаптивним кешем

Джерело даних може працювати навіть з неефективним методом кешу так, нібито кешу взагалі не існує.

Така архітектура інформаційної системи дозволяє реалізовувати окремі модулі окремо один від одного, лише підтримуючи інтерфейс взаємодії між окремими модулями. Реалізацію кожного окремого модуля можна провести на зручних для нього технологіях.

2.7. Висновки до розділу 2

У даному розділі було запропоновано модель адаптивного асоціативного кешу інформаційної системи. Визначені загальні характеристики та вимоги до інформаційної системи, яка може використовувати адаптивний модуль кешування.

Запропонована модель реалізації адаптивного кешу на основі визначених асоціативних правил з урахуванням. інформаційної системи. Визначено типові правила взаємодії в інформаційній системі за рахунок пошуку закономірностей у відправленні різних типів запитів для оптимального використання кешу. Типові правила, або асоціативні правила збереження даних до кешу визначаються на основі адаптації статистичних даних роботи інформаційної системи, історії запитів. Визначені правила заносяться в загальний стек асоціативних правил. При наявності набору таких асоціативних правил, система буде мати можливість розміщувати результати виконання ряду запитів в кеш ще до їх виникнення.

Описано та надано архітектуру інформаційної системи з асоціативним адаптивним кешем в своєму складі.

Надані загальні принципи побудови асоціативних правил та параметри роботи адаптивного модулю у складі адаптивного асоціативного кешу.

Надано опис превентивної функції асоціативних правил для покращення швидкодії інформаційної системи.

Описано алгоритми та особливості роботи інформаційної системи з адаптивним кешем у своєму складі.

3. ПРОГРАМНА РЕАЛІЗАЦІЯ МЕТОДУ

3.1. Вибір засобів реалізації

Для тестування, перевірки ефективності розробленого адаптивного кеша необхідно програмно реалізувати модель системи, що буде за структурою відповідати інформаційній системі, розглянутій у попередньому розділі.

3.1.1. *Вибір технологій для реалізації джерела даних*

Для створення програмного забезпечення необхідно створити джерело даних. Джерело даних доцільно реалізовувати на основі віддаленої бази даних так, як у інформаційних системах найчастіше саме база даних виступає головним джерелом даних.

Для створення джерела даних використовується реляційна СУБД щоб забезпечити стабільність, узгодженість і структурованість даних.

Серед реляційних СУБД доступними та поширеними для використання є MS SQL Server [15, 16], MySQL [17], Oracle [18, 19]. Кожна СУБД має свої переваги та недоліки, наведені в таблиці 5.

Таблиця 5

Порівняння СУБД

СУБД	Переваги	Недоліки
SQL Server	● Можливість стабільної роботи з великою кількістю даних. ● Ефективна реалізація запитів. ● Розвинений діалект T-SQL, який надає більші можливості обробки даних. ● Інтеграція з продуктами Microsoft.	● Складний процес установки, адміністрації та підтримання сервера. ● Потрібний більший апаратний ресурс. ● Складніше підключення стороннього ПЗ. ● Комерційне ПЗ.

Продовження табл. 5

MySQL	● Простий та швидкий процес налаштування та підтримання сервера. ● Зручне візуальне середовище відлагодження запитів. ● Наявність безкоштовної версії з повною функціональністю. ● Просте програмне підключення. ● Достатньо менш потужного сервера	● Не відповідає стандарту мови SQL. ● Менше документації та підтримки в разі проблем
Oracle	● Можливість стабільної роботи з великою кількістю даних. ● Ефективна реалізація запитів та допоміжних структур. ● Розвинена технологія, “перевірена часом”. ● Наявність безкоштовної версії.	● Дуже складний процес налаштування та підтримання сервера. ● Потрібен більш потужний сервер. ● Складне (орієнтоване на експерта) середовище відлагодження запитів ● Специфічний синтаксис запитів

З огляду на характеристики СУБД для створення системи було обрано MySQL, через простоту налаштування сервера БД та підключення до нього свого ПЗ та менші вимоги до апаратного забезпечення. База даних для використання в системі буде створюватись за необхідністю. Початкове тестування ефективності кешу буде проводитися на емуляторі запитів до бази, створеному власноруч.

3.1.2. *Вибір технологій для реалізації менеджера даних*

Необхідно обрати технології для реалізації менеджера даних:

● Для моніторингу роботи з даними.

- Для управління взаємодією між частинами інформаційної системи.
- Для конвертування даних для передачі інтерфейсу інформаційної системи.

Для створення менеджера даних використовується мова програмування, що дозволяє ефективно розробляти програмне забезпечення, яке розташоване на серверній частині. Також важливими вимогами до мови є наявність великої кількості вільних для використанні у науковому дослідженні бібліотек.

Серед мов програмування поширеними та вільними для використання у науковій розробці є наступні: Java [20], C# [21], C++ [22], Python [23]. Переваги та недоліки цих мов програмування наведені у таблиці 6.

Таблиця 6

Порівняння мов програмування

Мова програмування	Переваги	Недоліки
Java	● Середня швидкість розробки. ● Великий об’єм кодової бази, доступний у вільному доступі в інтернеті. ● Потужна та зручна середа розробки. ● Ефективна мова програмування. ● Легка у реалізації кросплатформенних рішень.	● Ресурсоємна середа роботи програми. ● Складне розгортання серверної частини системи. ● Великі вимоги до досвіду програмування на мові.

Продовження табл. 6

C#	● Інтеграція з продуктами Microsoft. ● Потужна та зручна середа розробки. ● Великий об’єм кодової бази, доступний у вільному доступі в інтернеті. ● Середня складність розробки серверної частини. ● Велика швидкість розробки.	● Складна у реалізації підтримки кросплатформності рішень. ● Ресурсоємна середа розробки. ● Ресурсоємна середа роботи коду ● Складне розгортання серверної частини системи.
C++	● Найшвидша і найефективніша мова програмування з розглянутих. ● Низьке споживання ресурсів серверу. ● Легка у реалізації кросплатформенних рішень.	● Складна у розробці мова програмування. ● Складна у розгортанні серверна частина. ● Великі вимоги до досвіду програмування для написання найефективніших рішень. ● Низька швидкість розробки.
Python	● Висока швидкість розробки. ● Велика кількість бібліотек у вільному доступі для використання. ● Легке розгортання серверної частини. ● Низькі вимоги до досвіду програмування для написання серверної частини. ● Легка у реалізації кросплатформенних рішень.	● Низька ефективність роботи програми.

З огляду на характеристики мов програмування та досвід розробника системи для створення менеджеру даних було обрано Python. Для менеджеру даних та для загальної моделі і перевірки ефективності кешування інформаційної системи не потрібна велика швидкість роботи серверної частини, тож кількість переваг використання Python для її написання є більшою за інші можливі для використання мови програмування.

3.1.3. *Вибір технологій для реалізації логу запитів*

Для зберігання історії викликів користувачів можна використовувати декілька підходів. Такий журнал транзакцій за своєю організацією подібний до зберігання логів та аналогічно працює з інформаційною системою. Логи можна зберігати за допомогою бази даних, файлів з різною структурою. У рамках даної роботи можна розглянути варіант зберігання у оперативній пам’яті так, як головною задачею є протестувати та довести наявність переваг у використанні певного методу кешування.

У таблиці 7 розглядаються можливі технології для реалізації журналу транзакцій з їх перевагами та недоліками.

Таблиця 7

Порівняння можливих місць зберігання журналу транзакцій

Місце зберігання журналу транзакцій	Переваги	Недоліки
База даних	● Простота зберігання запитів. ● Легкий процес структуризації запитів за допомогою реляційних таблиць.	● Необхідність розгортання бази даних. ● Додаткова витрата ресурсів на підтримку роботи бази даних.

Продовження табл. 7

	● Зрозумілий інтерфейс для взаємодії журналу транзакцій і менеджеру даних. ● Ефективні запити для отримання даних. ● Широкі можливості для фільтрації даних за допомогою запитів.	
Файл	● Значна кількість готових рішень для логування у файли. ● Легкий процес зберігання та переміщення статистичних даних.	● Неефективний процес отримання даних. ● Низькі можливості для фільтрації даних за запитами. ● Необхідність написання модулю що буде контролювати роботу з файлами журналу транзакцій.
Файлова база даних	● Наявність усіх переваг файлів та баз даних.	● Менша ефективність роботи запитів ніж з базою даних ● Обмежені можливості роботи з базами даних.
Оперативна пам’ять	● Зручність тестування та розробки.	● Неможливе використання у великих інформаційних системах із потужним навантаженням та кількістю запитів.

З огляду на характеристики можливих місць зберігання, досвід розробника системи для реалізації місця зберігання логу даних було обрано оперативну пам’ять. Таке рішення зумовлено тим, що для огляду та перевірки ефективності системи адаптивного асоціативного кешування система буде розгортатись на одному достатньо потужному пристрої. Ресурсів пристрою цілком достатньо для тестування ефективності

адаптивного асоціативного кешу із зберіганням логу запитів у оперативній пам’яті.

3.1.4. *Вибір технологій для реалізації асоціативного адаптивного модулю кешування*

Для реалізації асоціативного модулю можна використовувати мови програмування або спеціалізовані математичні пакети. У таблиці 8 наведені переваги та недоліки деяких математичних пакетів та мови програмування у якості технології реалізації адаптивного модулю.

Таблиця 8

Порівняння можливих технологій розробки адаптивного модулю кешування

Технологія розробки адаптивного модулю кешування	Переваги	Недоліки
Математичний пакет Maxima [24]	● Ефективний математичний пакет. ● Одна з найпотужніших систем візуалізації . ● Безкоштовне використання.	● Складність мови програмування ● Мала кількість туторіалів та готових рішень у вільному доступі.
Математичний пакет Matlab [25]	● Ефективний математичний пакет. ● Поширена кодова база, велика кількість готових рішень . ● Потужна система візуалізації. ● Легка мова програмування алгоритмів.	● Необхідна наявність ліцензії для використання.

Продовження табл. 8

Мова програмування	● Легка взаємодія з іншими модулями системи ● Просте рішення для розробки ● Проста система відладки та редагування алгоритмів ● Безкоштовне використання ● Можливість об’єднання з місцем зберігання кешу	● Менша ефективність ніж у математичних пакетах ● Необхідність написання окремого модуля для візуалізації результатів

З огляду на характеристики можливих місць зберігання, досвід розробника системи для реалізації асоціативного адаптивного модулю кешування була обрана мова програмування Python. Таке рішення також зумовлено тим, що усі варіанти кешу будуть розроблені у одній середі розробки, і можна бути певним у чесності порівняння результатів тестування ефективності рішення. Також місце зберігання кешу буде об’єднане з таким модулем, що простіше у реалізації.

3.1.5. *Вибір технологій для реалізації користувацького інтерфейсу*

Для реалізації користувацького інтерфейсу можна обрати будь-яку технологію що дозволяє реалізувати зручний у використанні інтерфейс, з якого можна посилати різноманітні запити на отримання інформації.

Слід зазначити, що у інформаційних системах реалізацій користувацького інтерфейсу може бути декілька, наприклад – для телефонів, планшетів, окремих операційних систем. Для тестування ефективності асоціативного адаптивного модулю достатньо реалізації одного наглядного для демонстрації прикладу користувацького інтерфейсу

інформаційної системи. У таблиці 9 наведені можливі для обрання технології з урахуванням досвіду розробника системи.

Таблиця 9

Порівняння можливих технологій розробки прикладу користувацького інтерфейсу до інформаційної системи

Технологія реалізації користувацького інтерфейсу	Переваги	Недоліки
JavaScript [26]	● Ефективні та прості у реалізації рішення ● Підтримка кросплатформності	● Низька безпека даних
C#	● Ефективні та прості у реалізації рішення	● Складна підтримка кросплатформності при використанні desktop-орієнтованих варіантів розробки
WxPython [27]	● Висока швидкість розробки ● Легка у реалізації кросплатформенних рішень	● Низька ефективність роботи програми

В зв'язку з тим, що безпека даних та кросплатформність не є цільовими при тестування варіантів розробки адаптивного асоціативного кешування, обрання технології можна залишити на перевагу розробника.

Було обрано Python та бібліотеку WxPython як легку для проєктування, розробки, підтримки інтерфейсу мову програмування.

3.2. Опис реалізації інформаційної системи з адаптивним модулем кешування

Реалізація та тестування ефективності асоціативного адаптивного модулю кешування є ключовою у даній роботі.

Відповідно до адаптивного асоціативного модулю кешування необхідно додати можливість перемикати різні види кешу для перевірки ефективності роботи різних типів алгоритмів кешування. Архітектура та взаємодії всередині модулю кешування наведені на рис. 11.

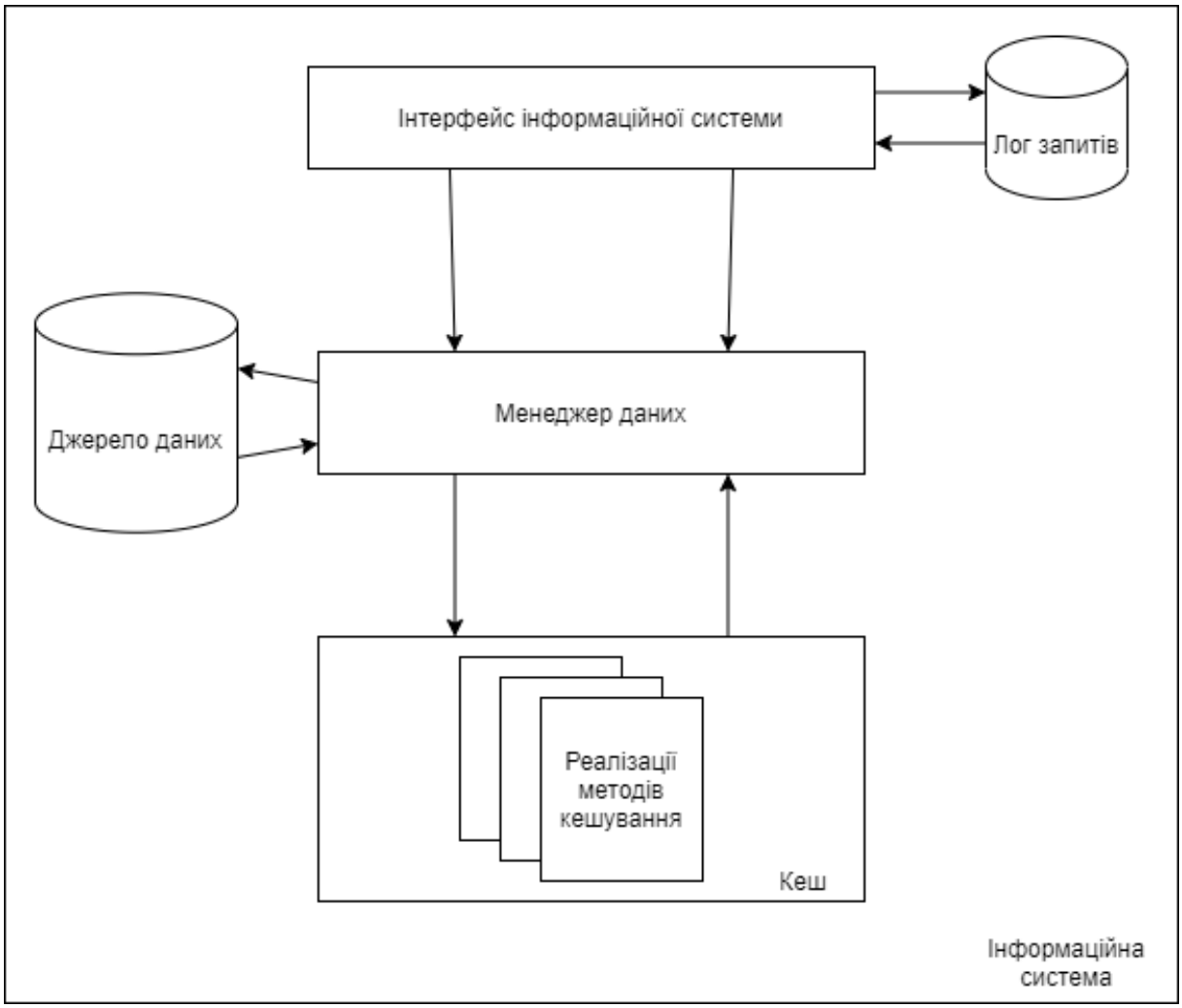


Рис. 11. Архітектура адаптивного модулю кешування

Інтерфейс інформаційної системи зберігає до логу запитів історію запитів. Якщо необхідно адаптувати систему відповідно до логу запитів

інтерфейс надсилає відповідний запит до менеджеру даних з логом запитів у якості параметру.

Менеджер даних може перемикати різні види кешу за запитом від інтерфейсу або відповідно до внутрішньої логіки. Менеджер даних перевіряє наявність даних усередині кешу, за необхідністю – зберігає дані до кешу. Якщо у кеші немає необхідних за запитом даних, менеджер даних звертається безпосередньо до джерела даних для отримання відповіді.

Кеш має декілька методів кешування всередині свого модулю, які можна перемикати за запитом зовні. Реалізація адаптивного асоціативного кешу містить асоціативні правила для адаптації роботи кешування.

Для використання асоціативних правил для рішення до збереження даних або предиктивного вибору даних необхідно, щоб була обрана реалізація адаптивного асоціативного кеша з загального набору методів кешування.

Адаптивний асоціативний модуль кешування отримує перебудовані правила з під-модулю перебудови асоціативних правил, який у свою чергу перебудовує їх за вимогою менеджера даних.

Модуль асоціативних правил для перебудови асоціативних правил використовує записи логу запитів для отримання статистичних даних використання системи.

3.3. Опис програмної реалізації макету інформаційної системи з адаптивним асоціативним кешем

Схема класів відображає програмну структуру інформаційної системи, розробленої у рамках роботи над дисертацією. Її графічне зображення можна побачити на рис. 12. Вона складається з наступних класів:

- InformationSystem – клас, що був побудований як макет інформаційної системи з гнучко розширюваним інтерфейсом.

- DataManager – клас, що був побудований як абстрактний менеджер даних, який може взаємодіяти та керувати з різними реалізаціями кешу та налаштовувати їх. Також він відповідає за надання відповіді інформаційній системі на запити відповідно до інтерфейсу.
- ICache – інтерфейс, що був побудований для надання абстрактного інтерфейсу кешу з яким буде взаємодіяти менеджер даних.

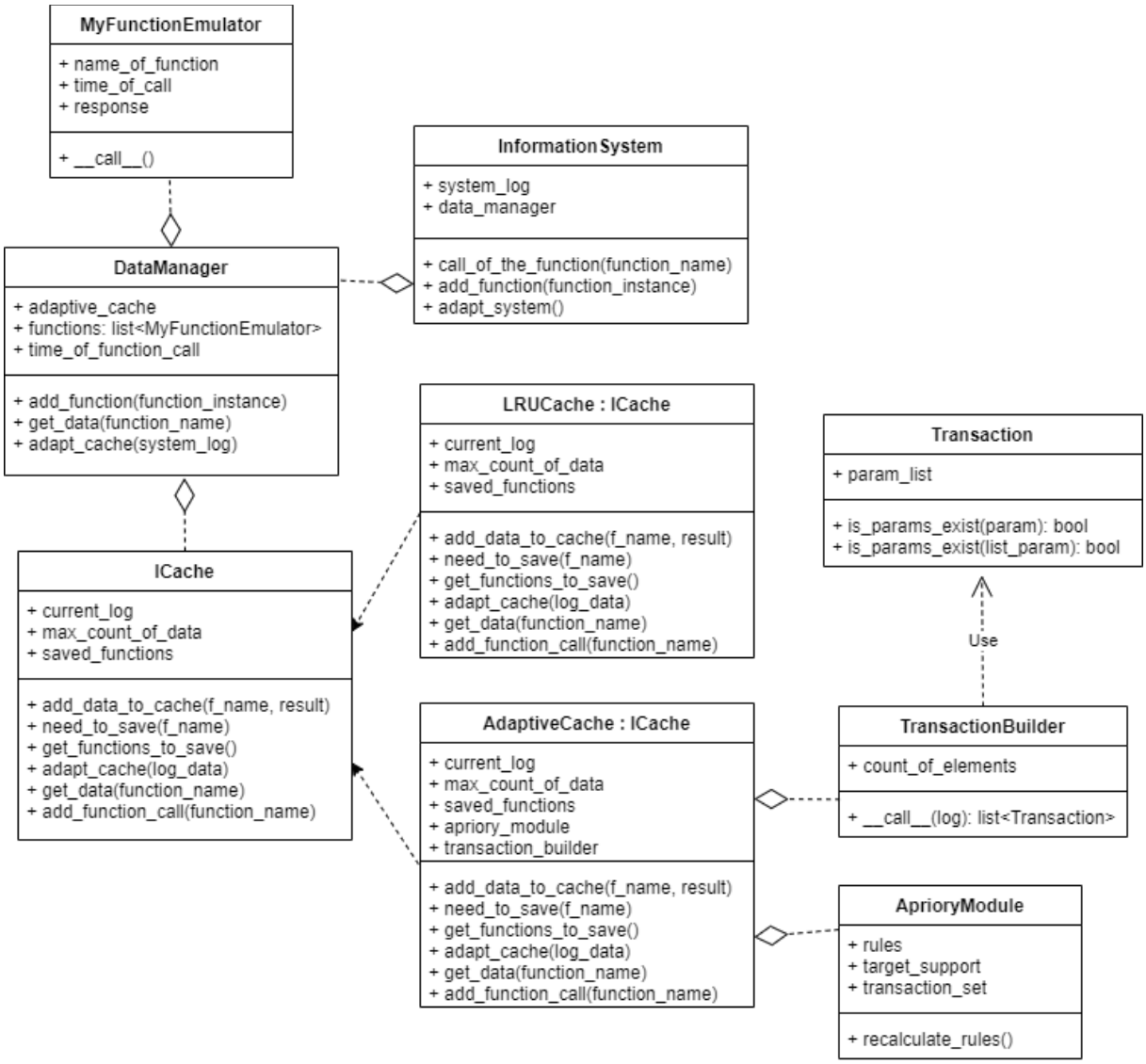


Рис. 12. Діаграма класів

- AdaptiveCache – клас-реалізація інтерфейсу ICache, що являє собою досліджуваний адаптивний асоціативний кеш

- LRUCache – клас-реалізація інтерфейсу ICache, що являє собою один з поширених методів кешування з якими порівнюється робота адаптивного асоціативного кешу
- ApriоryModule – клас, що реалізує алгоритм Apriory для побудови асоціативних правил та їх збереження.
- TransationBuilder – клас, що надає можливість з логу запитів створювати список транзакцій для побудови асоціативних правил.
- Transaction – клас-сутність транзакції, на основі списку яких створюються асоціативні правила.
- MyFunctionEmulator – клас, що був розроблений як емуляція елементу інтерфейсу інформаційної системи. Даний член інтерфейсу має відповідь на запит, власне ім’я, час виконання.

Детальніше з структурними елементами кожного класу можна ознайомитись у таблиці 10.

Таблиця 10

Опис класів

Назва класу	Елемент класу	Опис елементу
InformationSystem	system_log	Поле класу, що зберігає всередині лог викликів функцій інтерфейсу
	data_manager	Поле класу з реалізацією менеджеру даних для роботи інформаційної системи з даними
	call_of_the_function(name_of_function)	Функція для роботи з інтерфейсом системи. Як параметр приймає назву функції що буде викликана

Продовження табл. 10

	add_function(function_instance)	Функція для додавання нової функції до інтерфейсу системи. Як параметр приймає реалізацію функції яку можна буде викликати з методу call_of_the_function
	adapt_system()	Функція для адаптації системи. В даній реалізації викликає адаптацію кешу у DataManager
DataManager	adaptive_cache	Поле з реалізацією кешу, що імплементує інтерфейс ICache
	functions	Поле з списком функцій можливих для виклику. Це внутрішній інтерфейс запитів доступних у системі.
	time_of_function_call	Загальний час виконання функцій. Дане поле допомагає в оцінці ефективності системи кешування.
	add_function(function_instance)	Функція для додавання нової функції до списку функцій. Як параметр приймає реалізацію функції.
	adapt_cache(system_log)	Функція для адаптації кешу відповідно до логу який передається як параметр до функції адаптації кешу.

Продовження табл. 10

	get_data(function_name)	Функція для отримання результатів запиту. Як параметр приймає ім’я функції, результат якої необхідно надати у відповідь на запит інтерфейсу системи. Усередині функції менеджер даних перевіряє наявність даних у кеші та зберігає дані до кешу за необхідністю. Якщо даних у кеші немає – дані надаються як результат виклику функції.
ICache	current_log	Поле з логом, відповідно до якого була проведена остання адаптація кешу
	max_count_of_data	Максимальна кількість даних доступна для збереження у кеш
	saved_functions	Список з функціями, результат виконання яких був збережений до кешу
	add_data_to_cache(f_name ,result)	Функція для додавання даних до кешу. Як параметри приймає назву функції що була виконана та її результат
	need_to_save(f_name)	Функція що надає дані щодо необхідності збереження результату виконання запиту до кешу за назвою функції відповідно до поточного стану адаптації.

Продовження табл. 10

	get_functions_to_save()	Функція що надає список функцій що необхідно зберегти до кешу. Це механізм предиктивного отримання даних, відповідно до результатів виконання цієї функції до кешу можуть бути поміщені результати виконання запитів до безпосереднього запиту з боку інтерфейсу інформаційної системи.
	adapt_cache(log_data)	Функція адаптація кешу. Приймає лог запитів як параметр.
	get_data(f_name)	Функція надання даних за їх наявності у кешу відповідно до назви функції
	add_function_call(function_name)	Функція що додає запис до локальний логу нещодавно викликаних функцій. Це необхідно для підтримки механізму предиктивного додавання даних
AdaptiveCache	current_log	Поле з логом, відповідно до якого була проведена остання адаптація кешу
	max_count_of_data	Максимальна кількість даних доступна для збереження у адаптивний кеш
	saved_functions	Список з функціями, результат виконання яких був збережений до кешу

Продовження табл. 10

	apriory_module	Поле з реалізацією асоціативного модулю що будує асоціативні правила для адаптації кешу
	transaction_builder	Поле з реалізацією класу, що на основі логу запитів може створити список транзакцій
	add_data_to_cache(f_name, result)	Функція для додавання даних до кешу. Як параметри приймає назву функції що була виконана та її результат
	need_to_save()	Функція що надає дані щодо необхідності збереження результату виконання запиту до кешу за назвою функції відповідно до асоціативних правил наявних у асоціативному модулі.
	get_functions_to_save()	Функція що надає список функцій що необхідно зберегти до кешу. Надає список функцій, що будуть викликані з найбільшою ймовірністю відповідно до аналізу асоціативних правил та нещодавно викликаних функцій.
	get_data(f_name)	Функція, що надає дані відповідно до назви функції що передається як параметр.

Продовження табл. 10

	adapt_cache(log_data)	Функція, що виконує послідовність дій для побудови асоціативних правил відповідно до логу запитів, який надається як параметр. Будується список транзакцій, список транзакцій передається до асоціативного модулю, асоціативний модуль будує асоціативні правила які кеш може використовувати для адаптації своєї роботи
LRUCache	current_log	Поле з логом, відповідно до якого була проведена остання адаптація кешу
	max_count_of_data	Максимальна кількість даних доступна для збереження у кеш
	saved_functions	Список з функціями, результат виконання яких був збережений до кешу
	add_data_to_cache(f_name)	Функція для додавання даних до кешу. Як параметри приймає назву функції що була виконана та її результат
	need_to_save(f_name)	Функція що надає дані щодо необхідності збереження результату виконання запиту до кешу за назвою функції відповідно до методу роботи LRU
	get_functions_to_save()	LRU не має механізму предиктивного збереження, тому надається пустий список

Продовження табл. 10

	adapt_cache()	LRU не має спеціального механізму адаптації, тому ця функція не робить додаткових дій
	get_data()	Функція надання даних за їх наявності у кешу відповідно до назви функції
Transaction	param_list	Список параметрів які наявні усередині транзакції
	is_param_exist(param)	Додаткова функція перевірки наявності параметру усередині транзакції
	is_param_exist(list_param)	Додаткова функція перевірки наявності списку параметрів усередині транзакції
TransactionBuilder	count_of_elements	Параметр, що визначає кількість елементів всередині транзакції
	__call__(log)	Виклик побудови транзакцій, що надає список транзакцій у відповідь. Приймає лог запитів у вигляді списку
ApriotyModule	rules	Параметр з останнім побудованим списком правил
	target_Support	Параметр що визначає мінімальну підтримку для побудови правила
	transaction_set	Параметр з останнім завантаженим списком транзакцій

Продовження табл. 10

	recalculate_rules()	Функція що виконує побудову асоціативних правил відповідно до поточного списку транзакцій
MyFunctionEmulator	name_of_function	Ім’я функції
	time_of_call	Орієнтовний час виконання
	response	Результат виклику функції
	__call__()	Виклик функції, надає response у відповідь

3.4. Висновки до розділу 3

Для реалізації макету інформаційної системи для тестування адаптивного асоціативного модулю була обрана мова програмування Python. Для інтерфейсу користувача була обрана бібліотека PythonWx.

Для реалізації моделі адаптивного асоціативного кешування та менеджеру даних був також обраний Python для спрощення взаємодії між макетом інформаційної системи та модулем кешування, а також для можливості гнучко модифікувати налаштування між інтерфейсом, менеджером даних та адаптивним кешем.

Для емуляції роботи із базою даних був створений емулятор окремих запитів до бази даних, що можна налаштовувати власноруч. Це клас MyFunctionEmulator.

Був розроблений асоціативний адаптивний модуль кешування. Також був реалізований приклад інформаційної системи, що буде використовуватись для тестування ефективності розробленого методу кешування.

Розроблена інформаційна система має структуру, що дозволяє легко заміщати реалізації алгоритму кешування а також надає можливість збирання для подальшого аналіза роботи різних алгоритмів кешування.

Асоціативний модуль кешування використовує асоціативні правила для адаптації до характеру роботи системи, що відповідає меті наукової роботи.

Представлена розроблена система на рівні діаграми класів та проведений опис елементів розробленої інформаційної системи.

4. АНАЛІЗ ОТРИМАНИХ РЕЗУЛЬТАТІВ

4.1. Налаштування розробленої моделі інформаційної системи для роботи з різними типами кешування.

Розроблена модель інформаційної системи має структуру, яка дозволяє використовувати різні методи кешування у своєму складі. Для цього необхідно при ініціалізації менеджеру даних задати екземпляр класу того методу кешування, який необхідно обробити, протестувати та проаналізувати.

Це можна зробити як безпосередньо усередині Python-скрипта, так і обравши необхідний пункт у меню.

Скрипт із ініціалізацією методу кешування та обрання необхідного пункту у меню консолі можна побачити на лістингу 1 та рис. 13 відповідно.

Лістинг 1. Ініціалізація методу кешування в python скрипті

```
class InformationSystemEmulator:

    def __init__(self):
        self.system_log = list()
        self.data_manager = DataManager() # ініціалізація
менеджеру даних
        self.data_manager.adaptive_cache = LRUCache() #
Ініціалізація методу кешування в DataManager
```

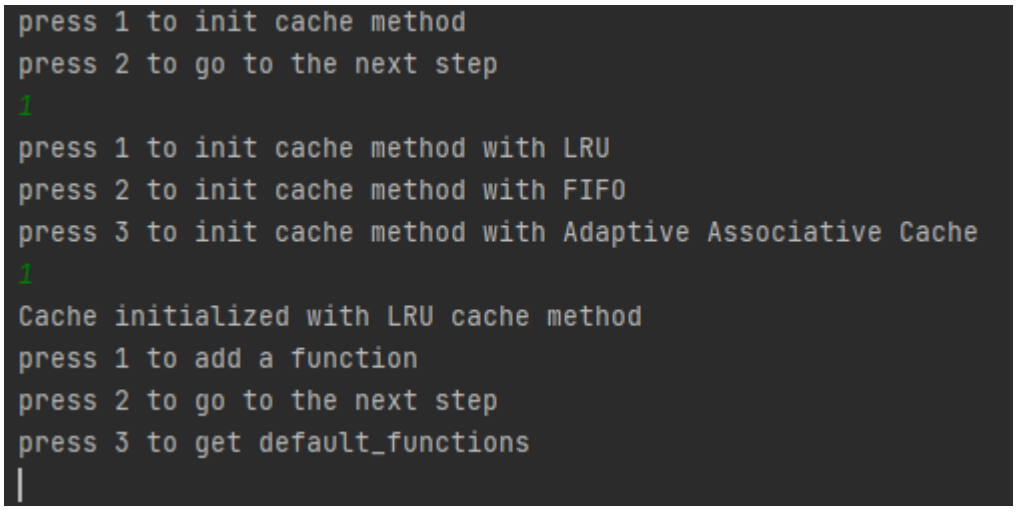


Рис. 13. Ініціалізація методу кешування з пункта меню

Завдяки однаковому інтерфейсу в різних реалізаціях методів кешування ніяких додаткових налаштувань не потрібно.

Використання кешу всередині менеджера даних не залежить від конкретної реалізації методу кешування. Головною умовою є наявність реалізації інтерфейсу ICache у класах кешування, екземплярами яких ініціалізується кеш всередині менеджера даних.

4.2. Огляд реалізації інформаційної системи

Інформаційна система була реалізована для тестування та аналізу роботи різних методів кешування. Для даної задачі інтерфейс інформаційної системи був розроблений з можливістю динамічно розширюватися. Відповідно до системи можна додавати функції, які будуть доступні для виклику з зовні. Ці функції відіграють роль запитів з боку інтерфейсу інформаційної системи до менеджеру даних.

Отже, інтерфейс інформаційної системи складається з функції для додавання інших функцій до менеджеру даних, функції для виклику динамічно заданих функцій, функції що викликає адаптацію системи шляхом надсилання запиту до менеджеру даних з логом запитів у якості параметра. Лог запитів поповнюється кожен раз, коли викликається функція. Цю функцію відображено на лістингу 2.

Лістинг 2. Функція виклику функції з списку доданих до системи функцій

```
def call_of_the_function(self, function_name):
     self.system_log.append(function_name)
     return self.data_manager.get_data(function_name)
```

4.3. Огляд реалізації менеджеру даних

Менеджер даних виконує функцію роботи з даними у інформаційній системі. Він надає можливість абстрагуватись від джерела даних та обробляти запити з боку інтерфейсу системи.

В поточній реалізації інформаційної системи були додані додаткові можливості до інтерфейсу та менеджеру, для спрощення тестування та аналізу роботи інформаційної системи з різними методами кешування.

До менеджеру даних було додано можливість динамічно розширювати інтерфейс системи шляхом додавання нових функцій, які може викликати інтерфейс. Функцію додавання елементу до інтерфейсу інформаційної системи відображено у лістингу 3.

Лістинг 3. Додавання функції до загального списку функцій що може викликати інтерфейс з менеджера даних

```
def add_function(self, function_instance):
    self.data_manager.add_function(function_instance)
```

Також до менеджеру даних додано лічильник часу виконання функцій. Цей лічильник рахує час, який інтерфейс очікує отримання відповіді на свій запит. Таким чином, чим менше значення лічильника часу, тим ефективніше працює кеш системи.

Функція отримання даних з менеджеру даних складається з декількох етапів. Дану функцію можна побачити на лістингу 4.

Спочатку до екземпляру кешу додається інформація щодо виклику функції. Ця дія додається до кроків виконання запиту у менеджер даних для того, щоб підтримати механізм предиктивного додавання даних в кеш до запиту. Далі з кешу отримується список функцій, які з великою долею ймовірності можуть бути викликані у найближчий час. Відповідно результат виконання даних функцій предиктивно додається до кешу.

Наступним кроком перевіряється наявність даних всередині кешу. Якщо дані знаходяться всередині кешу, то вони повертаються як результат виконання запиту.

У іншому випадку до лічильника часу додається час виконання функції, а результат виконання функції повертається у відповідь на запит.

Також менеджер даних запитує у кешу необхідність додавання цих даних до кешу, і при отриманні ствердної відповіді дані додаються до кешу.

Лістинг 4. Функція отримання даних з менеджеру даних

```
   def get_data(self, function_name):
        self.adaptive_cache.add_function_call(function_name)
        predicted_functions =
self.adaptive_cache.get_functions_to_save()
        for f in predicted_functions:
            self.adaptive_cache.add_data_to_cache(f,
self.get_function(f)())
        if function_name in
self.adaptive_cache.saved_functions:
            return self.adaptive_cache.get_data(function_name)
        self.time_of_functions_call +=
self.get_function(function_name).time_of_call
        result = self.get_function(function_name)()
        if self.adaptive_cache.need_to_save(function_name):

self.adaptive_cache.add_data_to_cache(function_name, result)
        return result
```

Менеджер даних не прив’язується до конкретної реалізації кешу, тому адаптація кешу проводиться за логікою, інкапсульованої у конкретній реалізації кешу в методі adapt_cache. Функція виклику адаптації кешу можна побачити у лістингу 5. Як параметр до кешу передається лог запитів, за яким кеш може адаптувати свою роботу.

Лістинг 5. Функція отримання даних з менеджеру даних

```
   def adapt_cache(self, system_log):
        self.adaptive_cache.adapt_cache(system_log)
```

4.4. Огляд реалізації розробленого модулю адаптивного асоціативного кешу

Адаптивний асоціативний кеш реалізований для використання асоціативних правил для прийняття рішень видалення та додавання даних. Адаптація адаптивного асоціативного кешу проводиться шляхом переведення логу запитів до списку транзакцій та знаходження асоціативних правил на їх основі. Функції адаптації кешу у лістингу 6.

Лістинг 6. Функції адаптації кешу

```
   def adapt_cache(self, log_data):
        self.current_log = log_data
        self.apriory_module.transaction_set =
self.get_transactions()
        self.apriory_module.recalculate_rules()

    def get_transactions(self):
        return self.transaction_builder(self.current_log)
```

Для знаходження асоціативних правил використовується апріорний модуль, в якому знаходиться реалізація апріорного алгоритму знаходження асоціативних правил. За необхідності видалення елементу з кешу, обирається елемент з найменшою долею підтримки у асоціативних правилах.

Додавання даних до кешу складається з декількох кроків. Спочатку перевіряється наявність вільного місця у кеші. За його відсутністю викликається функція видалення елементів з кешу.

Далі до кешу зберігаються дані, запит на збереження яких був отриманий ззовні.

Адаптивний асоціативний модуль кешування має власну реалізацію методу, що повертає функції які необхідно заздалегідь зберегти до кешу. Функція повернення спрогнозованих запитів зображена на лістингу 7. Аналізуючи останні викликані функції, список яких поповнюється ззовні, кеш повертає ті функції, які з найбільшою долею ймовірності будуть запитані.

Лістинг 7. Функція, що повертає функції пов’язані з нещодавно викликаними функціями

```
def get_functions_to_save(self):
     for rule in self.apriory_module.rules:
         if len(rule[0]) == 1: continue
         check = True
         for fun in self.called_function:
             if fun not in rule[0]:
                 check = False
                 break
         if check:
```

```
                functions_to_save = list()
                for f in rule[0]:
                    if f not in self.called_function:
                        functions_to_save.append(f)
                return functions_to_save

        return list()
```

4.5. Огляд реалізації розробленого модулю пошуку асоціативних правил

Реалізація модулю пошуку асоціативних правил грунтується на апріорному алгоритмі пошуку асоціативних правил. Він параметризується за кількома параметрами. Параметри можна побачити у лістингу 8.

Лістинг 8. Параметри реалізації апріорного алгоритму знаходження асоціативних правил

```
class AprioryModule:
    def __init__(self):
        self.param_support={}
        self.transaction_set = []
        self.target_support = 0.3
        self.min_param_support = 0.5
        self.rules = []
```

В даній реалізації алгоритм параметризований за мінімальною підтримкою для правил. Цей параметр відповідає за граничне значення підтримки, при якому правила вважаються валідними та додаються до списку правил.

Список транзакцій – ще один параметр. За цим списком знаходяться асоціативні правила при виклику функції перебудови правил.

Відмінність реалізованого алгоритму від звичайного апріорного алгоритму є те, що до списку правил додаються також окремі параметри. Тобто створюються правила, що складаються з єдиного елемента. Мінімальна підтримка таких елементів для їх додавання задається за додатковим параметром мінімальної підтримки параметра – min_param_support.

Така відмінність зумовлена необхідністю аналізувати необхідність додання до кешу результатів запитів, не пов’язаних з іншими запитами.

Програмно знаходження асоціативних правил реалізовано за такими кроками:

1. Знаходження усіх запитів, що задовольняють за значенням підтримки значенню target_support.
   1.1. Додавання до списку асоціативних правил запитів, значення підтримки яких задовольняють значенню min_param_support.
2. Створення кандидатів у асоціативні правила з пар запитів.
3. Перевірка кандидатів на їх значення підтримки у списку транзакцій. Додавання до асоціативних правил кандидатів що пройшли перевірку на необхідну підтримку. Якщо не додано нових асоціативних правил, перейти до кроку 5.
4. Створення нових кандидатів з асоціативних правил, що пройшли перевірку на значення підтримки шляхом додавання нових запитів до їх складу. Перехід до кроку 3.
5. Завершення роботи алгоритму.

4.6. Огляд реалізації модулю розбиття логів на транзакції

Реалізація модулю розбиття логів на транзакції грунтується на розбитті логів на рівні транзакції за кількостю. До конструктора класу модулю надається максимальна кількість елементів в транзакції, при виклику модулю із логами у якості параметру як результат віддається набір транзакції за наданими логами.

Реалізацію розбиття на транзакції списку логів можна побачити у лістингу 9.

Лістинг 9. Функція розбиття логів на транзакції

```
def __call__(self, log):
    transactions = list()
    length_of_transaction = 0
```

```
        current_transaction = Transaction()
        for i in range(len(log)):
            current_transaction.param_dictionary[log[i]] =
log[i]
            length_of_transaction += 1
            if current_transaction not in transactions:
                transactions.append(current_transaction)
            if length_of_transaction >=
self.count_of_elements_n_trasaction:
                current_transaction = Transaction()
                length_of_transaction = 0

        return transactions
```

4.7. Порівняння роботи адаптивного асоціативного кешу з іншими методами роботи кешу

Адаптивний асоціативний кеш має можливість адаптуватися до характеру використання системи відповідно до логу запитів, що зберігається при кожному запиті до менеджеру даних. Це є головною відмінністю адаптивного асоціативного кешу від інших методів кешування. Алгоритми кешування, оглянуті раніше у даній роботі адаптуються під систему у поточному режимі. Наприклад, LFU зберігає інформацію щодо кількості викликів кожної функції всередині себе і поповнює цю кількість при кожному запиті. Але у великих системах зазвичай це не має сенсу, так як відносна кількість запитів все одно залишається незмінною.

Адаптивний асоціативний кеш може налаштовуватись у довільні проміжки часу. Доцільно використати періоди простою системи для налаштування та адаптації кешу до закономірностей використання системи. Таким чином адаптація кешу не буде негативно впливати на швидкість роботи системи з-за додаткового навантаження.

У результаті роботи інформаційної системи можна отримати результати тестування адаптивного асоціативного кешу.

4.8. Тестування та аналіз роботи адаптивного асоціативного кешу у порівнянні з іншими методами роботи кешу

Інформаційна система, спроєктована та реалізована у роботі запускається у декількох режимах отримання інформації. Результати та їх аналіз можна побачити у табл. 11.

Таблиця 11

Тестування та аналіз роботи методів кешування

Метод кешування	Ефективність при випадкових запитах	Ефективність при наявності послідовностей у запитах	Простота реалізації	Додаткове навантаження системи	Можливості налаштування
FIFO	3	1	10	1	1
LRU	3	5	8	4	4
Адаптивний асоціативний кеш	3	9	2	5	10

У першому режимі роботи з інформацією дані з джерела витягуються абсолютно хаотично і без закономірностей.

Аналіз результатів тестування асоціативного адаптивного кешу показує, що при хаотичній роботі з системою асоціативний адаптивний кеш не дає ніяких переваг перед іншими типами кешування. Але він не програє, так як до початку адаптації а також при відсутності знайдених закономірностей доцільно використовувати звичайний метод кешування на основі існуючих. Такий підхід також є частиною адаптації кешу до роботи системи.

У другому режимі звертання до інформаційної системи вже мають деяку структуру. Тобто у наборі викликів функції можна відслідкувати певну закономірність.

Результатом тестування асоціативного адаптивного кешу у таких вимогах є наступне. При великої кількості наявних закономірностей кеш видає набагато кращі результати ніж такі алгоритми кешування як FIFO, LFU, LRU (таблиця 11), але слід зазначити, що на початку роботи системи, коли журнал транзакцій пустий – асоціативний адаптивний кеш працює на одному рівні за ефективністю, що і інші алгоритми кешування.

Із порівняльної таблиці можна побачити, що додаткове навантаження системи з боку такого модулю кешування незначне, але адаптивний асоціативний кеш дає великий приріст у ефективності роботи коли у запитах можна знайти певні закономірності, а відповідно – і асоціативні правила.

Також до порівняння був доданий стовпець з можливостями налаштуваннями алгоритмів. Цей стовпець відображає можливості налаштування алгоритму на інформаційну систему. Адаптивний асоціативний метод досить ефективний так, як велику кількість параметрів роботи даного алгоритму можна налаштовувати з оглядом на ознаки системи. Також можливе введення додаткових наборів асоціативних правил, наприклад: правила заборони на видалення, що будуть слідкувати за наявними зв’язками елементів, які видаляються з кешу та іншими збереженими до кешу даними.

Параметри роботи адаптивного асоціативного алгоритму залежать від загальної кількості запитів збережених в системі, максимального об’єму кешування, характеристики потужності системи.

4.9. Особливості роботи створеної моделі та програмної реалізації

Для отримання переваги роботи асоціативного адаптивного кешу визначилося доцільним використання додаткового методу кешування – обираючи з стандартних алгоритмів. Використання стандартного алгоритму припиняється, коли під час чергової адаптації асоціативного кешу знаходиться достатня кількість асоціативних правил для використання даних правил для роботи кешу.

Визначення вдалого моменту для припинення використання стандартного алгоритму кешування є складною алгоритмічною задачею. У даній роботі це визначається за допомогою заздалегідь заданих параметрів кешування, тобто визначення необхідної кількості запитів у логу, запитів для аналізу.

4.10. Напрямки вдосконалення та подальша робота

Під час виконання роботи були виявленні наступні напрямки вдосконалення та місця для дослідження.

Розбиття логу запитів. Інформаційні системи містять системи логування запитів, параметрами яких є ідентифікатори користувачів що надсилають запити на отримання інформації. Тож цілком логічно розбивати журнал з збереженими логами викликів за користувачами і адаптувати роботу кешу під кожного користувача окремо. Для цього вдалим рішенням є використання розподілених баз даних, таких як Redis – як структурного елементу для збереження кешу. Необхідно розробити модель системи з використанням розподілених баз даних як місця знаходження кешу та перевірити ефективність вдосконалення таких систем з використанням асоціативних адаптивних кешей.

Також задача розбиття журналу викликів на транзакції не є тривіальною. У роботі розбиття виконується за заданим алгоритмом – обираються проміжки часу, або кількість елементів у транзакції, і виклики об’єднуються у транзакції. Але слід зазначити, що це один з найпростіших варіантів розбиття транзакцій. Знаходження ефективних способів розбиття загального списку запитів на транзакції є одним з напрямків вдосконалення адаптивного асоціативного методу кешування.

4.11. Висновки до розділу 4

В даному розділі був розглянутий процес налаштування розробленої інформаційної системи та менеджеру кешування для роботи із різними методами кешування.

Оглянута та описана реалізація інформаційної системи для тестування та порівняння різних методів кешування.

Розглянута робота менеджеру даних та особливості розробленого програмного забезпечення для роботи з даними. Описана взаємодія менеджеру даних та інтерфейсу системи, елементи керування кешем із боку менеджеру даних.

Описані особливості розробленого модулю адаптивного кешування. Оглянуто використання асоціативних правил всередині модулю адаптивного кешування для реалізації адаптивного асоціативного кешу всередині даного модулю. Також розглянутий модуль пошуку асоціативних правил та описані модифікації внесені до нього для спрощення роботи кешу із модулем побудови асоціативних правил.

Було проведено порівняння роботи адаптивного асоціативного методу кешування із іншими методами роботи кешу. Описані суттєві відмінності роботи адаптивного асоціативного кешу від роботи інших розглянутих методів кешування.

У даному розділу були розглянуті результати тестування моделі асоціативного адаптивного кешу у інформаційних системах. Були виділені переваги та недоліки використання адаптивного кешу та умови його ефективної роботи.

Розглянуті особливості роботи інформаційної системи, що розглядається та умови тестування спроєктованої системи з використанням різних типів кешування.

Були розглянуті шляхи подальшого вдосконалення алгоритмічної частини моделі асоціативного адаптивного кешу, розглянуті можливості для

вдосконалення структури інформаційної системи на якій проводиться дослідження.

ВИСНОВКИ

Метою даної магістерської дисертації було дослідження розробленого метода адаптивного кешування інформаційної системи із асоціативними правилами в своєму складі, які будуються на основі статистичних даних роботи інформаційної системи. Такий адаптивний кеш надасть можливість покращити швидкодію інформаційної системи завдяки адаптації кешування до правил використання системою, що можуть бути знайдені та проаналізовані з логів використання системи.

Розглянуто проблему ефективності кешування в інформаційних системах. Розглянуто необхідність створення нового методу адаптивного асоціативного кешу для покращення ефективності роботи кешу в інформаційних системах. Розглянуто існуюче програмне забезпечення для кешування в інформаційних системах, оглянуто та проаналізовано поширені методи кешування, виявлені загальні недоліки існуючих методів кешування. Виявлена можливість адаптувати систему кешування відповідно до існуючого логу запитів в інформаційній системі.

Запропоновано нову модель адаптивного асоціативного кешу для використання в інформаційній системі. Визначені характеристики та особливості роботи цільової для адаптивного асоціативного кешу інформаційної системи. Описана модель реалізації адаптивного кешу на основі знаходження взаємозв’язків між запитами в історії запитів збереженої в логу запитів. Такі взаємозв’язки знаходяться з використанням математичних алгоритмів пошуку асоціативних правил в наборі транзакцій. Надано архітектуру інформаційної системи, що ефективно може використовувати адаптивний асоціативний кеш в своєму складі для підвищення ефективності роботи інформаційної системи. Виявлена додаткова можливість використання асоціативних правил в адаптивному асоціативному кеші, а саме – превентивного додавання інформації до кешу

до безпосереднього запиту. Це досягається шляхом прогнозування наступних запитів відповідно до поточних.

Розроблена модель інформаційної системи з модулем кешування в своєму складі. В рамках розробленої моделі надана можливість змінювати реалізацію методу кешування. Описана структура розробленої моделі інформаційної системи. Реалізована модель адаптивного асоціативного кешу. Описана структура роботи адаптивного асоціативного кешу. Розглянуті особливості використання асоціативних правил всередині модулю адаптивного кешування. Зроблено порівняння роботи інформаційної системи з адаптивним асоціативним кешем в своєму складі та декількома іншими методами кешування.

СПИСОК ВИКОРИСТАНИХ ЛІТЕРАТУРНИХ ДЖЕРЕЛ

1. Коваленко, О. Є. Стандартизація формального опису системної архітектури [Електронний ресурс] Інститут проблем математичних машин і систем НАН України. – 2015. – Режим доступу до ресурсу: http://conf.atsukr.org.ua/conf_files/conf_dir_24/Kovalenko_sppr2015.pdf.
2. Tanenbaum, A. S. Modern operating systems./ Tanenbaum, A. S., Bos, H. // – Pearson, 2015.
3. LRU Cache [Електронний ресурс] – Режим доступу до ресурсу: https://www.interviewcake.com/concept/java/lru-cache.
4. Dar, S., Semantic data caching and replacement / Dar, S., Franklin, M. J., Jonsson, B. T., Srivastava, D., & Tan, M. //VLDB. – 1996. – Т. 96. – С. 330-341.
5. Maffeis, S. Cache management algorithms for flexible filesystems //ACM SIGMETRICS Performance Evaluation Review. – 1993. – Т. 21. – №. 2. – С. 16-25.
6. Karedla, R., Caching strategies to improve disk system performance / Karedla R., Love J. S., Wherry B. G. //Computer. – 1994. – Т. 27. – №. 3. – С. 38-46.
7. Megiddo, N. ARC: A Self-Tuning, Low Overhead Replacement Cache / Megiddo N., Modha D. S.//Fast. – 2003. – Т. 3. – №. 2003. – С. 115-130.
8. Okhrymchuk D. D. Integer Norm for Difference Assessment of the Frame Elements Considering the White Balance /Ivashchenko M. V., Okhrymchuk D. D., Lyushenko L. A/ Управляющие системы и машины. – 2019.
9. What is Memcached? [Електронний ресурс] – Режим доступу до ресурсу: http://memcached.org/
10. Redis. [Електронний ресурс] – Режим доступу до ресурсу: https://redis.io/

11. Django’s cache framework. [Електронний ресурс] – Режим доступу до ресурсу: https://docs.djangoproject.com/en/3.2/topics/cache/
12. Hornik, K. A computational environment for mining association rules and frequent item sets /Hornik K., Grün B., Hahsler M.//Journal of statistical software. – 2005. – Т. 14. – №. 15. – С. 1-25.
13. Agrawal, R. Fast algorithms for mining association rules / Agrawal, R., Srikant, R. //Proc. 20th int. conf. very large data bases, VLDB. – 1994. – Т. 1215. – С. 487-499.
14. Agrawal, R. Fast discovery of association rules / Agrawal, R., Mannila, H., Srikant, R., Toivonen, H., & Verkamo, A. I. //Advances in knowledge discovery and data mining. – 1996. – Т. 12. – №. 1. – С. 307-328.
15. Руководство по MS SQL Server 2019. [Електронний ресурс] – Режим доступу до ресурсу: https://metanit.com/sql/sqlserver/
16. Петкович, Д. Microsoft SQL Server 2012. Руководство для начинающих //СПб.: БХВ-Петербург. – 2013.
17. Введення в MySQL. Установка сервера. [Електронний ресурс]. – 2018. – Режим доступу до ресурсу: https://metanit.com/sql/mysql/1.1.php.
18. Introduction to the Oracle Database. [Електронний ресурс]. – 2019. – Режим доступу до ресурсу: https://docs.oracle.com/cd/B13789_01/server.101/b10743/intro.htm.
19. Advantages & Disadvantages of Oracle SQL. [Електронний ресурс]. – 2015. – Режим доступу до ресурсу: https://www.techwalla.com/articles/advantages-disadvantages-of-oraclesql.
20. Schildt, H. Java: the complete reference. / Schildt H., Coward D. //New York : McGraw-Hill Education, 2014. – С. 1312.
21. Шилдт Г. Полный справочник по C#/Г //Шилдт,–М.: Издательский дом “Вильямс.” – 2004.
22. Страуструп Б. Программирование. Принципы и практика использования C++. – Litres, 2021.

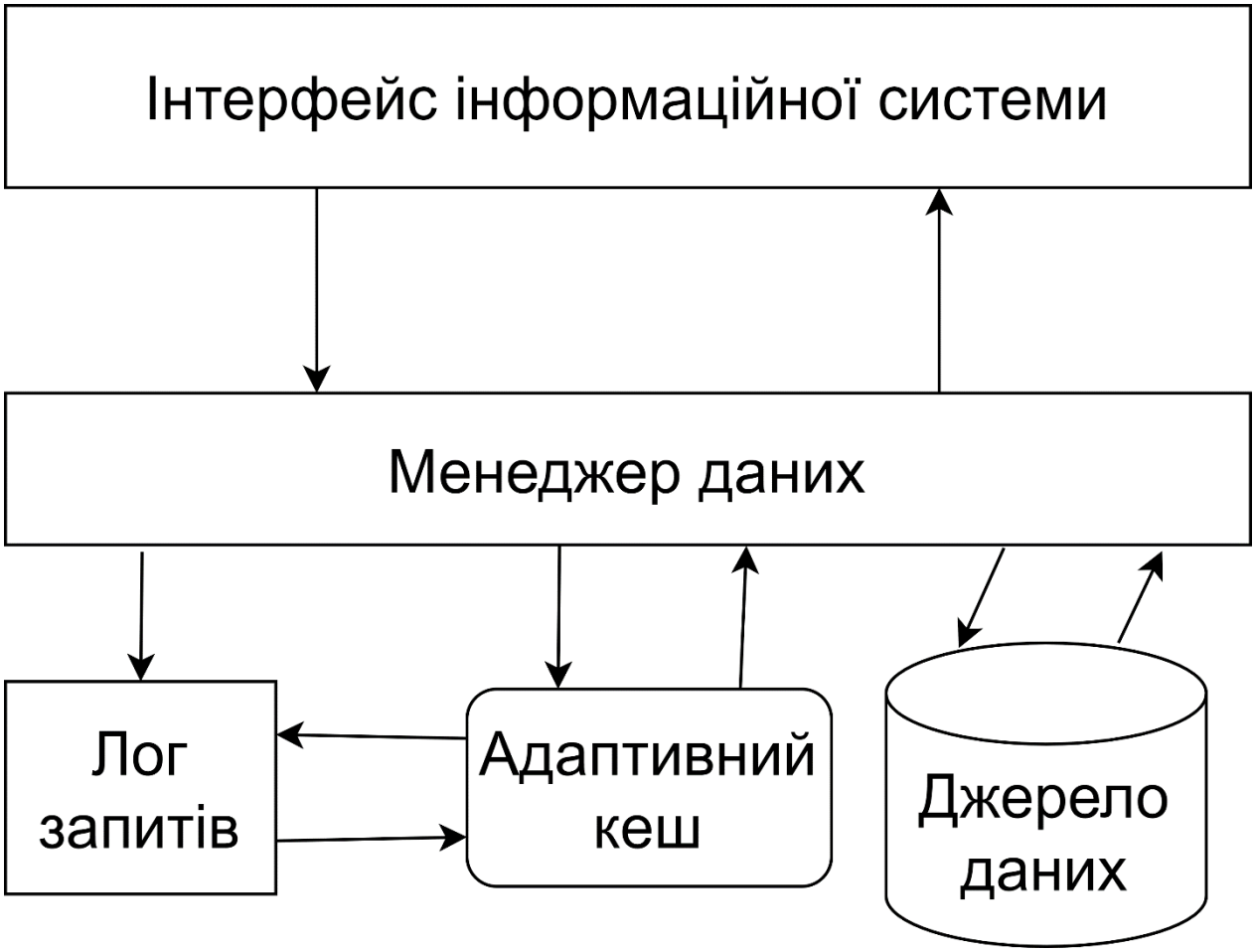
23. Рамальо Л. Python. К вершинам мастерства. – Litres, 2019.
24. Система компьютерной алгебры Maxima. [Електронний ресурс] – Режим доступу до ресурсу: https://maxima.sourceforge.io/ru/
25. MatLab. [Електронний ресурс] – Режим доступу до ресурсу: https://compress.ru/article.aspx?id=16152#MatLab
26. White A. Major JavaScript Engines //JavaScript Programmer's Reference.—Indianapolis, IN. – Т. 46256. – С. 12-13.
27. Welcome to wxPython. [Електронний ресурс] – Режим доступу до ресурсу: https://www.wxpython.org/

ДОДАТКИ

Додаток 1

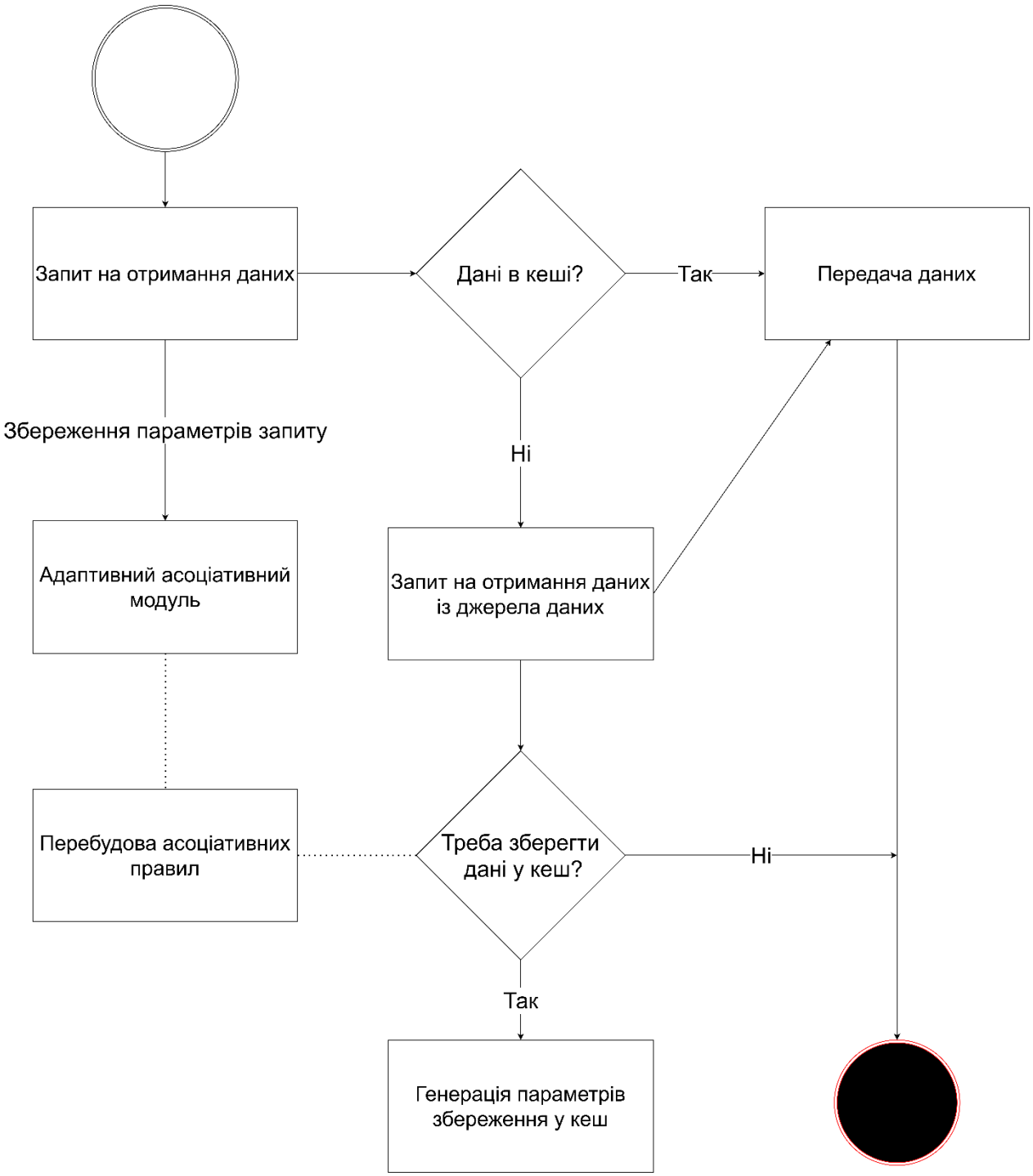
Копії графічних матеріалів

Загальна модель системи з адаптивним асоціативним кешем



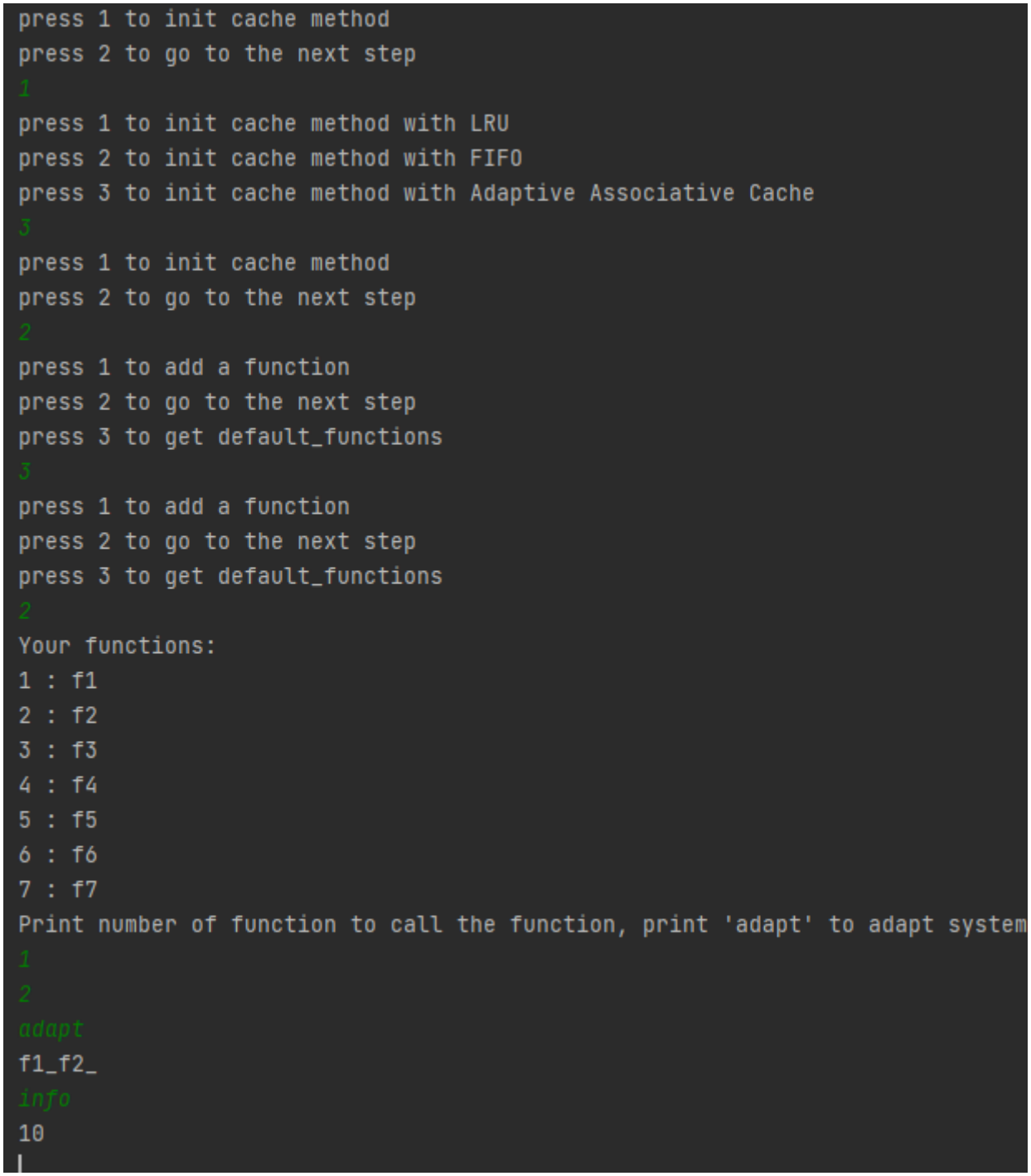
Охримчук Д. Д., КП-91мн

Метод адаптивного асоціативного кешування



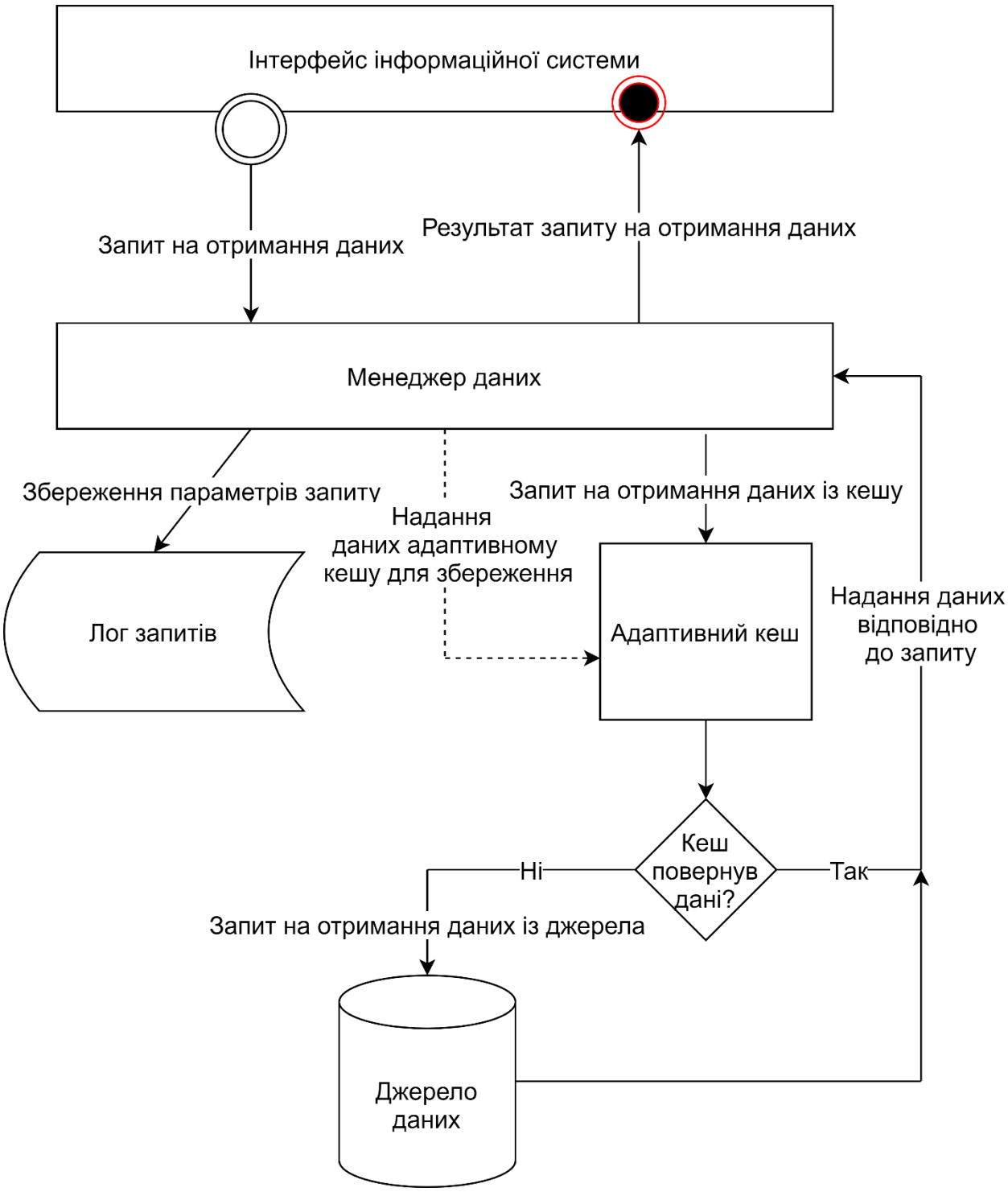
Охримчук Д. Д., КП-91мн

Інтерфейс розробленої моделі інформаційної системи



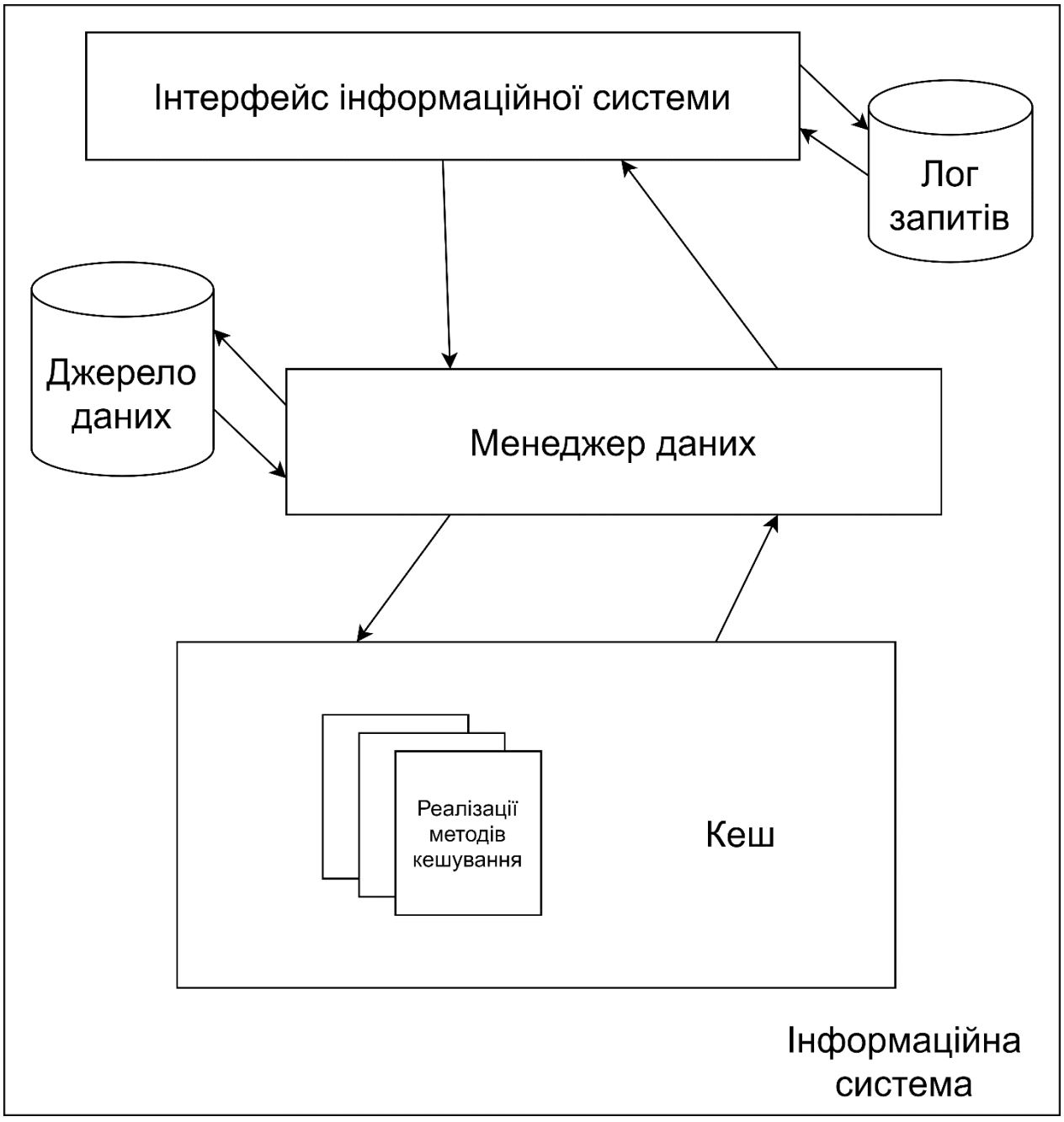
Охримчук Д. Д., КП-91мн

Архітектура системі із адаптивним асоціативним кешем



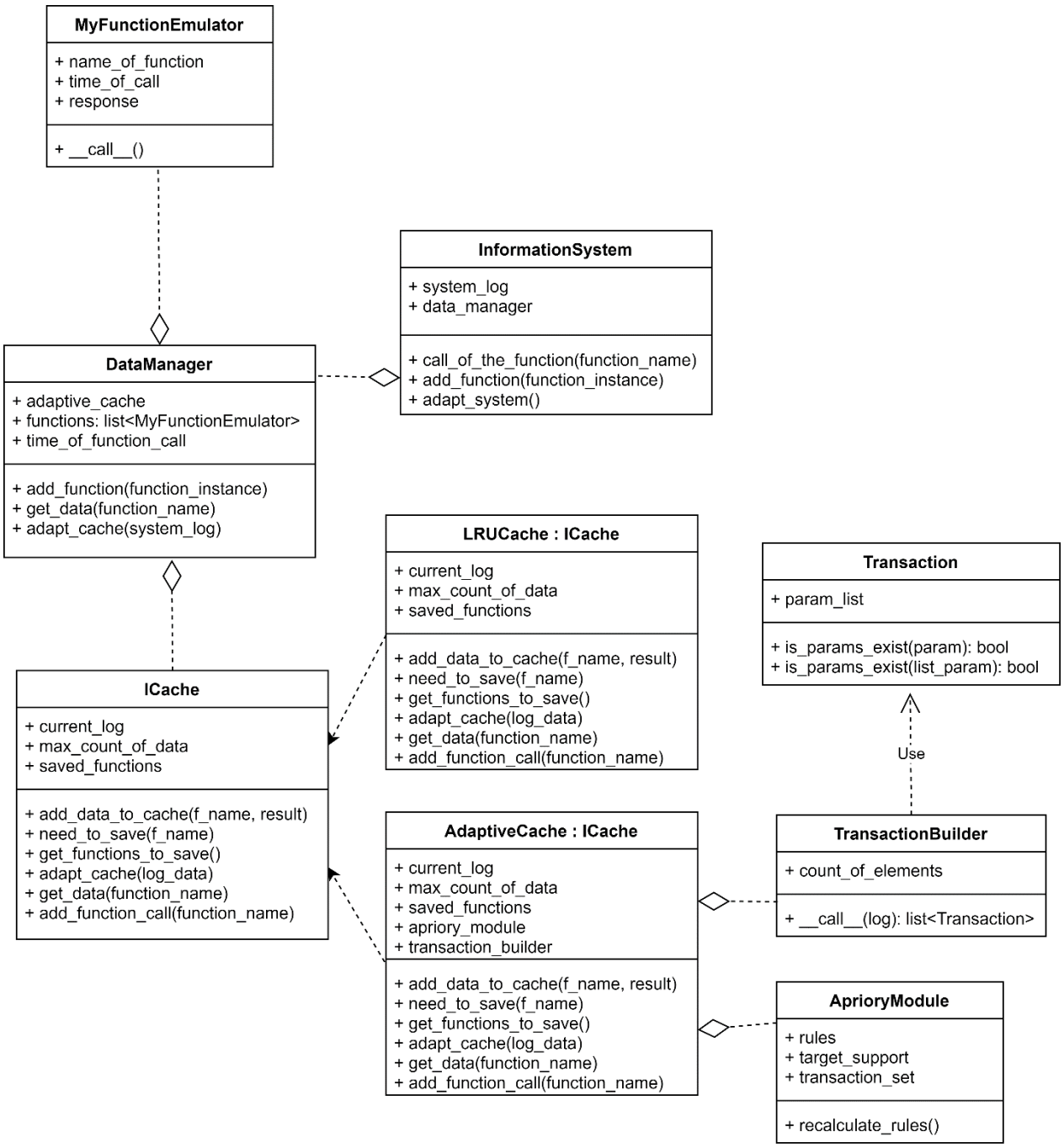
Охримчук Д. Д., КП-91мн

Архітектура інформаційної системи із кількома реалізаціями методів кешування



Охримчук Д. Д., КП-91мн

Діаграма класів розробленої моделі інформаційної системи



Охримчук Д. Д., КП-91мн

Додаток 2

Лістинг програми

Лістинг 1. Модуль побудови асоціативних правил

```
from MyTransaction import Transaction


class Candidate:
    def __init__(self):
        self.min_index_in_supported_params=0
        self.items=[]

    def __deepcopy__(self, memo):
        new_candidate = Candidate()
        new_candidate.min_index_in_supported_params =
self.min_index_in_supported_params
        for item in self.items:
            new_candidate.items.append(item)
        return new_candidate

    def __str__(self):
        str =""
        for item in self.items:
            str+=item.__str__() +"_"
        return str


class AprioryModule:
    def __init__(self):
        self.param_support={}
        self.transaction_set = []
        self.target_support = 0.3
        self.min_param_support = 0.5
        self.rules = []

    def __calculate_support_for_rule(self, rule):
        count_of_match = 0
        for transaction in self.transaction_set:
            if transaction.is_params_exist(rule):
                count_of_match+=1
        return count_of_match / len(self.transaction_set)

    def recalculate_rules(self):
        self.rules = []
        supported_params_set = []
        self.param_support={}

        for transaction in self.transaction_set:
            for key in transaction.param_dictionary:
                if not key in self.param_support.keys():
                    self.param_support[key]=1
                else:
                    self.param_support[key]+=1

        for key in self.param_support:
            param_support =
self.param_support[key]/len(self.transaction_set)
            if param_support >= self.min_param_support:
```

```
                supported_params_set.append(key)
                candidate = Candidate()
                candidate.items.append(key)
                self.rules.append((candidate.items, param_support))

        candidates=[]

        for i in range(0, len(supported_params_set)):
            for j in range(i+1, len(supported_params_set)):
                candidate = Candidate()
                candidate.min_index_in_supported_params = j+1
                candidate.items.append(supported_params_set[i])
                candidate.items.append(supported_params_set[j])
                candidates.append(candidate)
                #print(candidate.__str__()+"  :
"+candidate.min_index_in_supported_params.__str__())

        index_of_candidate_to_check = 0

        '''
        ways of impoving:
        recurtion
        delete candidate after checking
        '''

        while(index_of_candidate_to_check < len(candidates)):
            support_for_rule =
self.__calculate_support_for_rule(candidates[index_of_candidate_to_c
heck].items)
            if support_for_rule >= self.target_support:

self.rules.append((candidates[index_of_candidate_to_check].items,
support_for_rule))
                print(candidates[index_of_candidate_to_check])
                for i in range(

candidates[index_of_candidate_to_check].min_index_in_supported_param
s,
                len(supported_params_set)):
                    new_candidate =
candidates[index_of_candidate_to_check].__deepcopy__(None)

new_candidate.items.append(supported_params_set[i])
                    new_candidate.min_index_in_supported_params =
i+1
                    candidates.append(new_candidate)
                    #print(new_candidate.__str__()+"  :::
"+candidate.min_index_in_supported_params.__str__())
            index_of_candidate_to_check+=1
```

Лістинг 2. Модель інформаційної системи

```
import time
from MyTransaction import Transaction
from AprioryAlg import AprioryModule
from MyFunctionEmulator import MyFunctionEmulator
from TransactionBuilder import TransactionBuilder
from LRUCache import LRUCache
from AdaptiveCache import AdaptiveCache
from DataManager import DataManager

class InformationSystemEmulator:

    def __init__(self):
        self.system_log = list()
        self.data_manager = DataManager() # ініціалізація менеджеру
даних
        #self.data_manager.adaptive_cache = LRUCache() #
Ініціалізація методу кешування в DataManager

    def call_of_the_function(self, function_name):
        self.system_log.append(function_name)
        return self.data_manager.get_data(function_name)

    def add_function(self, function_instance):
        self.data_manager.add_function(function_instance)

    def adapt_system(self):
        self.data_manager.adapt_cache(self.system_log)
```

Додаток 3

Копія презентації

НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
“КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ ІМЕНІ ІГОРЯ СІКОРСЬКОГО”

ФАКУЛЬТЕТ ПРИКЛАДНОЇ МАТЕМАТИКИ

КАФЕДРА ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ КОМП’ЮТЕРНИХ СИСТЕМ

Метод побудови моделі адаптивного асоціативного кешу в інформаційних системах

Науковий керівник: доцент кафедри ПЗКС, к.т.н., Люшенко Л. А.

Доповідач: Охримчук Денис

Київ – 2020

Мета. Завдання.

- Мета дослідження: підвищення ефективності роботи систем кешування в інформаційних системах шляхом розроблення адаптивного асоціативного модулю кешування.
- Завдання дослідження:
  - Розглянути методи кешування
  - Дослідити переваги та недоліки існуючих методів кешування
  - Розробити модель кешування що буде мати переваги над існуючими аналогами із використанням адаптивних алгоритмів

Предмет дослідження

Предметом дослідження є:

- Методи роботи кешу
- Способи адаптації моделей кешування
- Алгоритми побудови асоціативних правил

Постановка задачі

Задано інформаційну систему з модулем зберігання статистики.

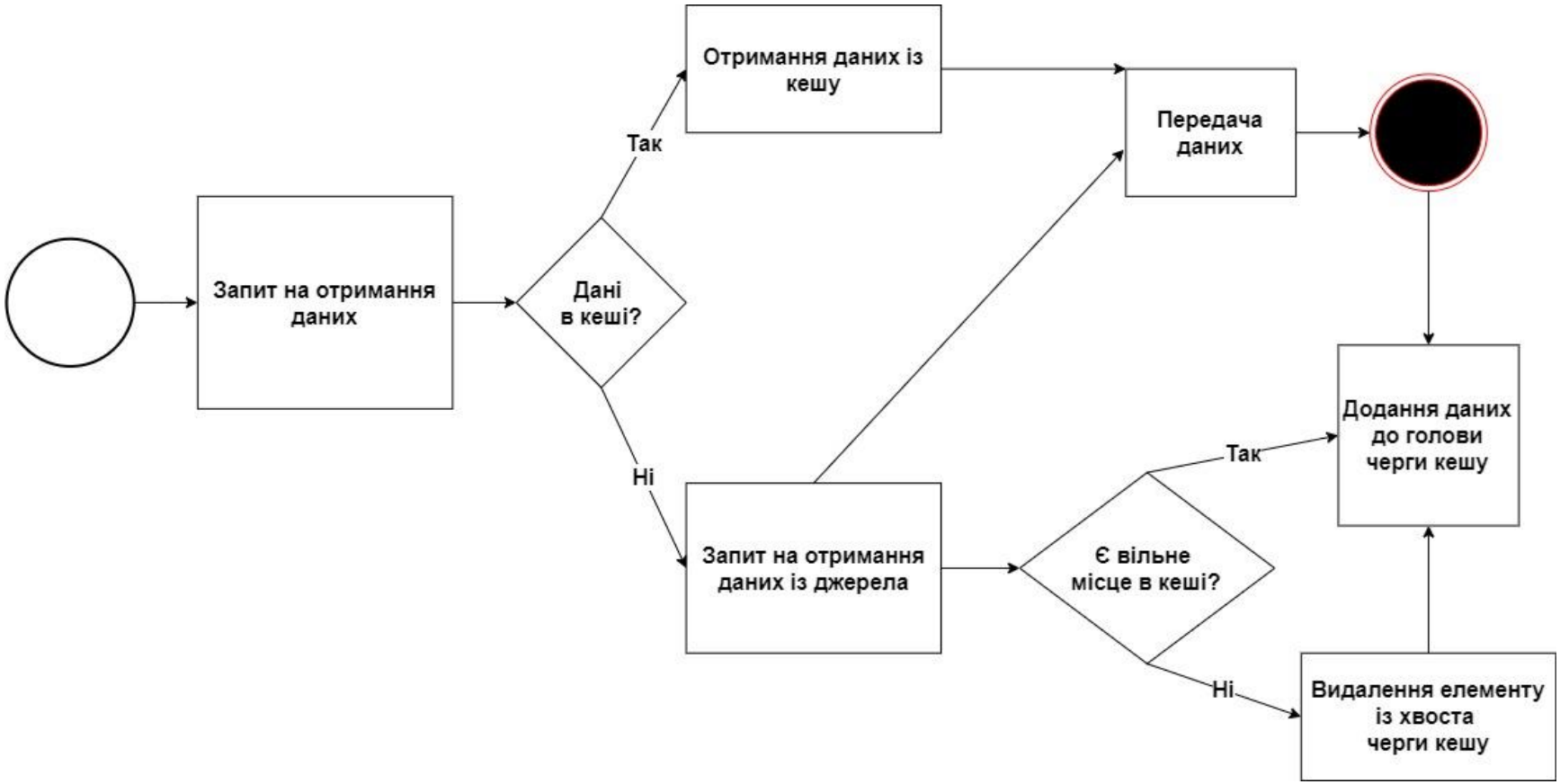
Необхідно побудувати ефективну модель адаптивного кешування. Важливо використовувати статистичні дані для адаптації алгоритму кешування до інформаційної системи.

Така модель кешування має позитивно вплинути на ефективність роботи кешування у більшості інформаційних системах.

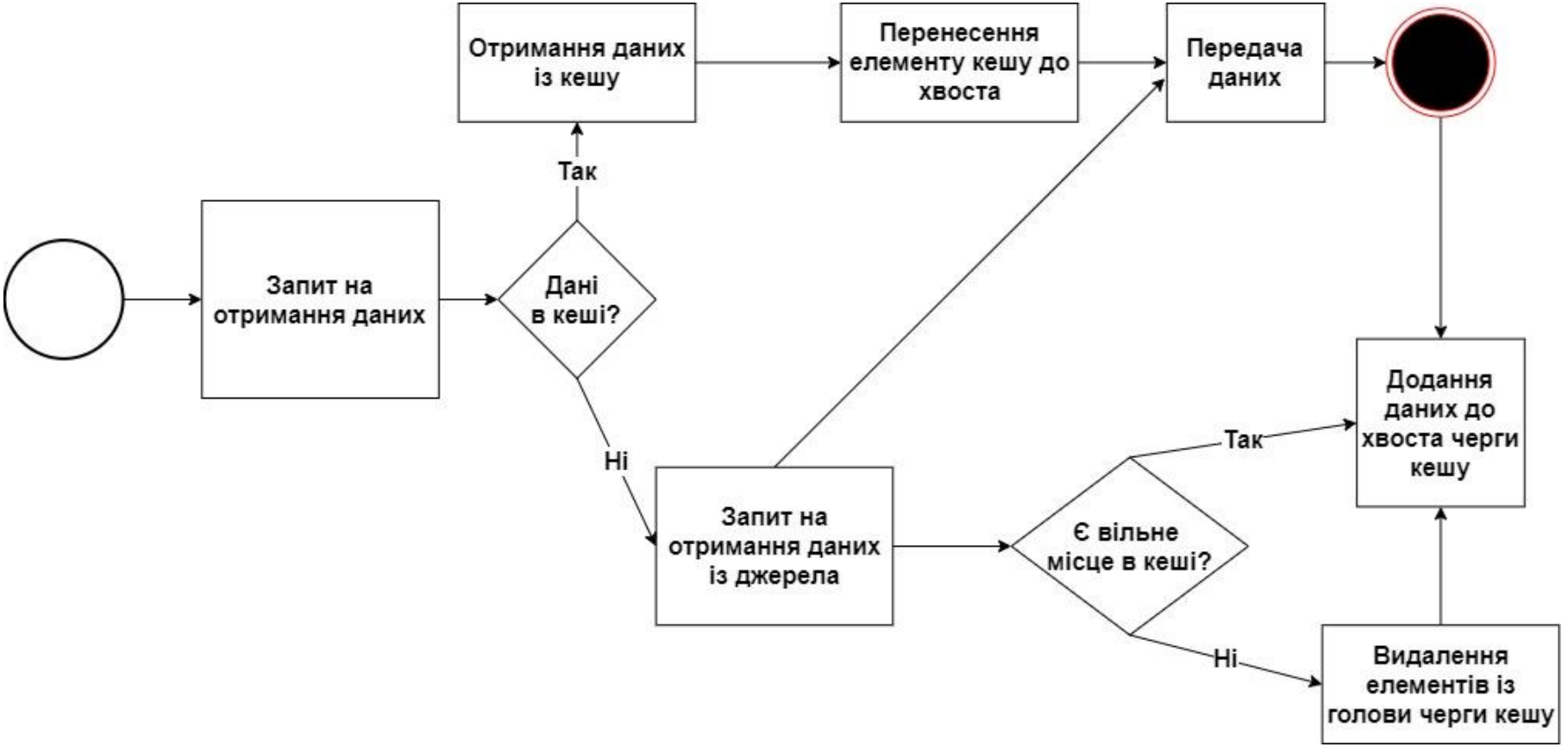
Огляд методів кешування

- FIFO - First In First Out.
- LFU - Least Recently Used.
- LRU - Least Frequency Used.
- SN LRU - Segmented LRU.

FIFO


Запит на отримання даних
Дані в кеші?
Так
Отримання даних із кешу
Ні
Запит на отримання даних із джерела
Передача даних
Є вільне місце в кеші?
Так
Додання даних до голови черги кешу
Ні
Видалення елементу із хвоста черги кешу

LRU


Отримання даних із кешу
Перенесення елементу кешу до хвоста
Передача даних
Так
Запит на отримання даних
Дані в кеші?
Додання даних до хвоста черги кешу
Ні
Так
Запит на отримання даних із джерела
Є вільне місце в кеші?
Ні
Видалення елементів із голови черги кешу

LFU

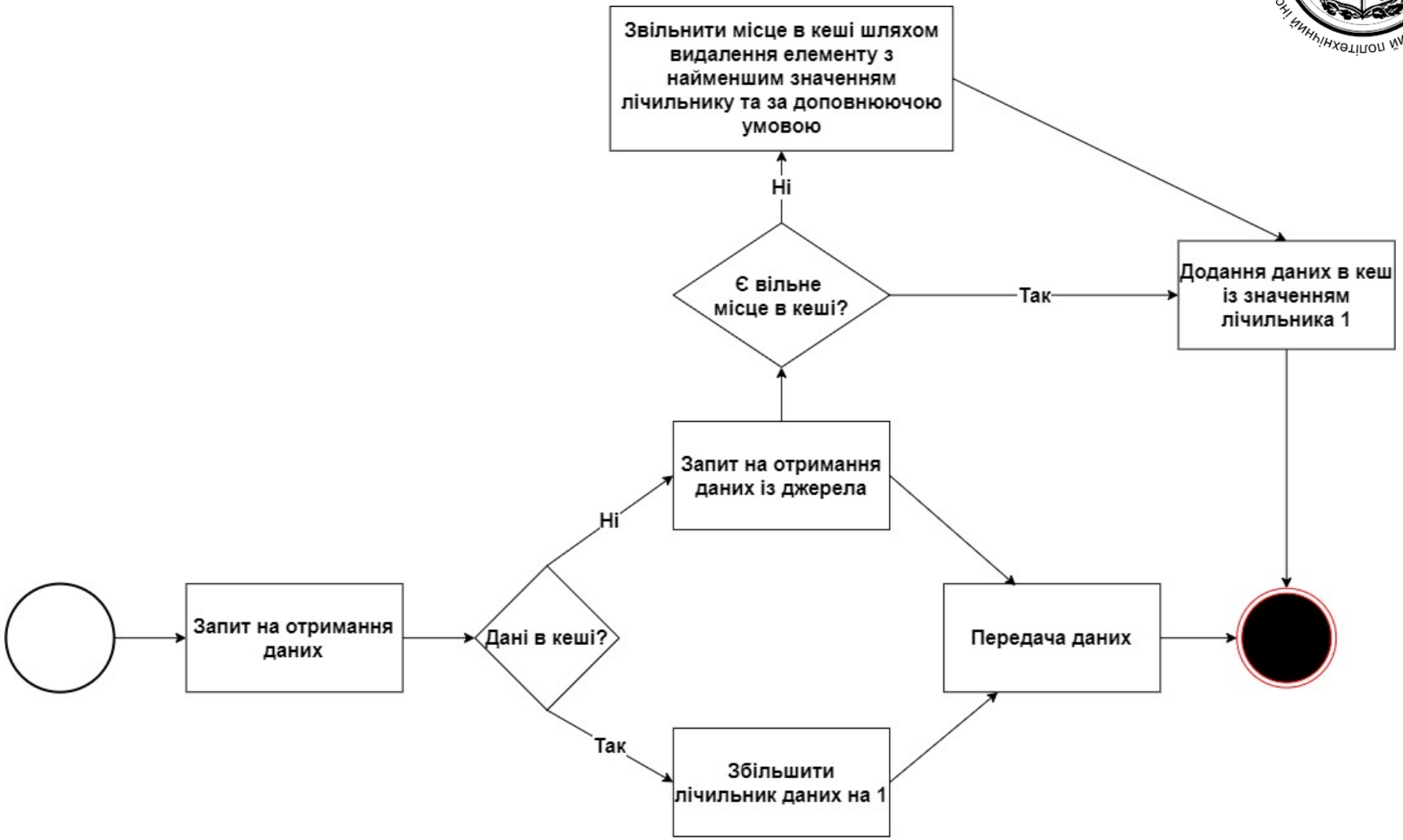

Звільнити місце в кеші шляхом видалення елементу з найменшим значенням лічильнику та за доповнюючою умовою
Ні
Є вільне місце в кеші?
Так
Додання даних в кеш із значенням лічильника 1
Запит на отримання даних із джерела
Ні
Запит на отримання даних
Дані в кеші?
Передача даних
Так
Збільшити лічильник даних на 1

Схема LFU кешу

Лічильник	Закешовані дані
5	info 1
4	info 2
...	...
k1	info i
k2	info i2

SN LRU

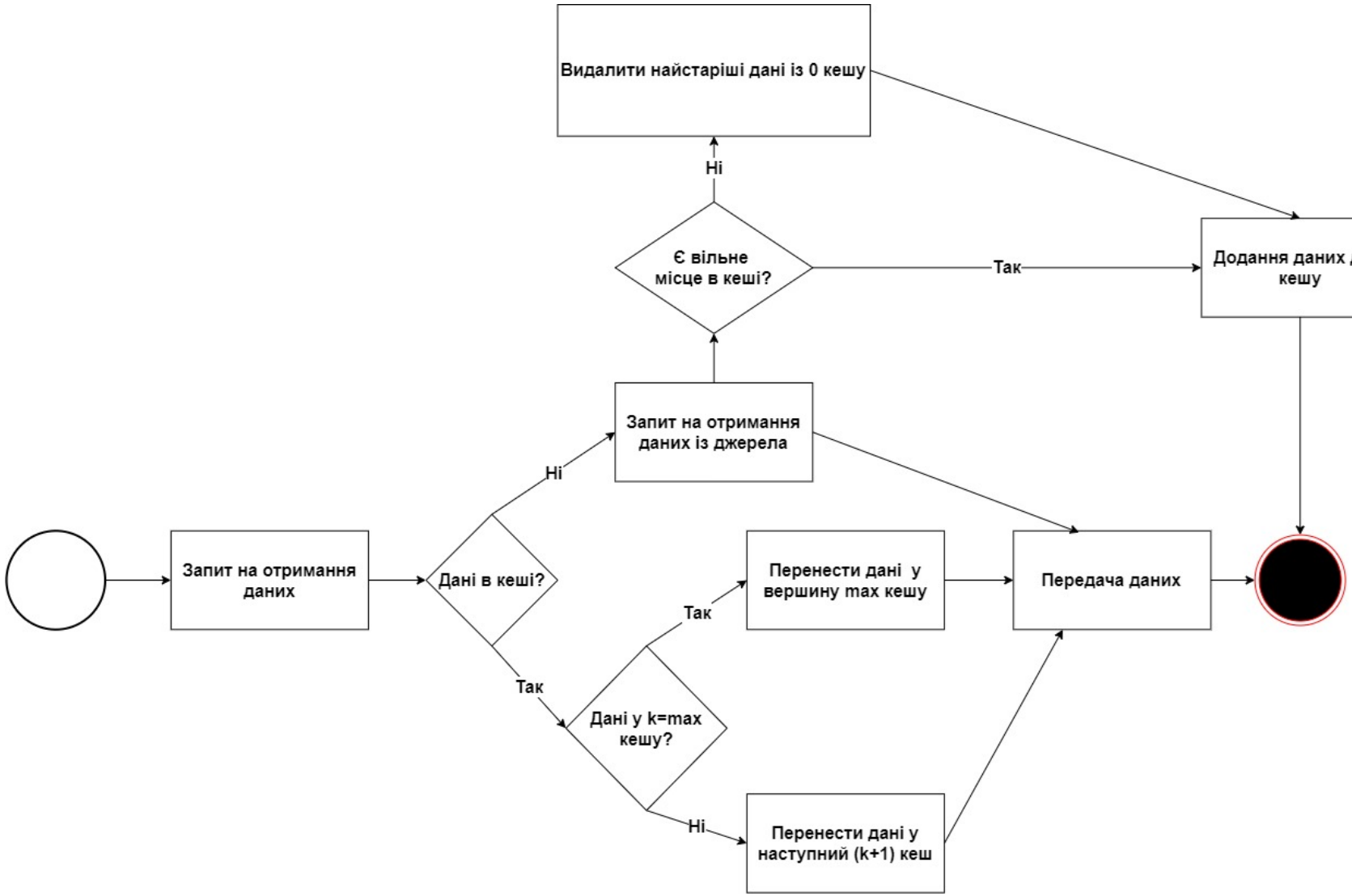

Видалити найстаріші дані із 0 кешу
Ні
Є вільне місце в кеші?
Так
Додання даних до 0 кешу
Запит на отримання даних із джерела
Ні
Запит на отримання даних
Дані в кеші?
Так
Перенести дані у вершину max кешу
Передача даних
Так
Дані у k=max кешу?
Ні
Перенести дані у наступний (k+1) кеш

Схема SN LRU

Недоліки існуючих алгоритмів

- Розглянуті методи кешування не використовують можливостей аналізу логів роботи інформаційної системи, що можуть стати потужним інструментом для адаптації кешування до інформаційної системи. Відповідно робота кешу в інформаційних системах може бути покращена шляхом використання логу запитів для адаптації роботи кешу

Модифікація алгоритму кешування

- Заміна стандартного методу кешування на адаптивний асоціативний метод кешування.
- Використання статистичних даних, отриманих під час роботи системи для підтримки адаптивності методу кешування
- Для побудови адаптивного асоціативного методу кешування використовуються математичні алгоритми побудови асоціативних правил

Асоціативні правила. Приклад

Номер транзакції	Елемент 1	Елемент 2	Елемент 3	Елемент 4
1	1	0	1	1
2	0	0	0	1
3	1	0	1	0
4	0	1	1	1
5	1	1	1	1
6	1	1	1	0

Асоціативні правила. Приклад

Номер транзакції	Елемент 1	Елемент 2	Елемент 3	Елемент 4
1	1	0	1	1
2	0	0	0	1
3	1	0	1	0
4	0	1	1	1
5	1	1	1	1
6	1	1	1	0

Асоціативні правила. Характеристики

X-набір Query, який досліджується на наявність правила. | D (X) | - кількість транзакцій, в яких є даний набір queries, | T | - загальна кількість транзакцій.

Support - це частотний показник досліджуваного набору значень параметрів в розглянутому наборі транзакцій.

$$supp(X) = \frac{|D(X)|}{|T|}$$

Confidence - це достовірність досліджуваного твердження, що визначає його вірогідність.

Приклад твердження: якщо x=A, то y= B (x, y - параметри, A та B - їх значення).

$$conf(X) = \frac{sup(X)}{sup(qi,\ qi\ C\ X)}$$

Lift - характеристика, що слугує величиною вимірювання залежності одного правила від іншого.

$$lift(qi \cup qj) = \frac{supp(qi \cup qj)}{supp(qi) * supp(qj)}$$

Conviction - це значення, що визначає частоту "відсутності спрацьовування" правила, що розглядається.

$$conv(qi \cup qj) = \frac{1 - supp(qj)}{1 - conf(qi \cup qj)}$$

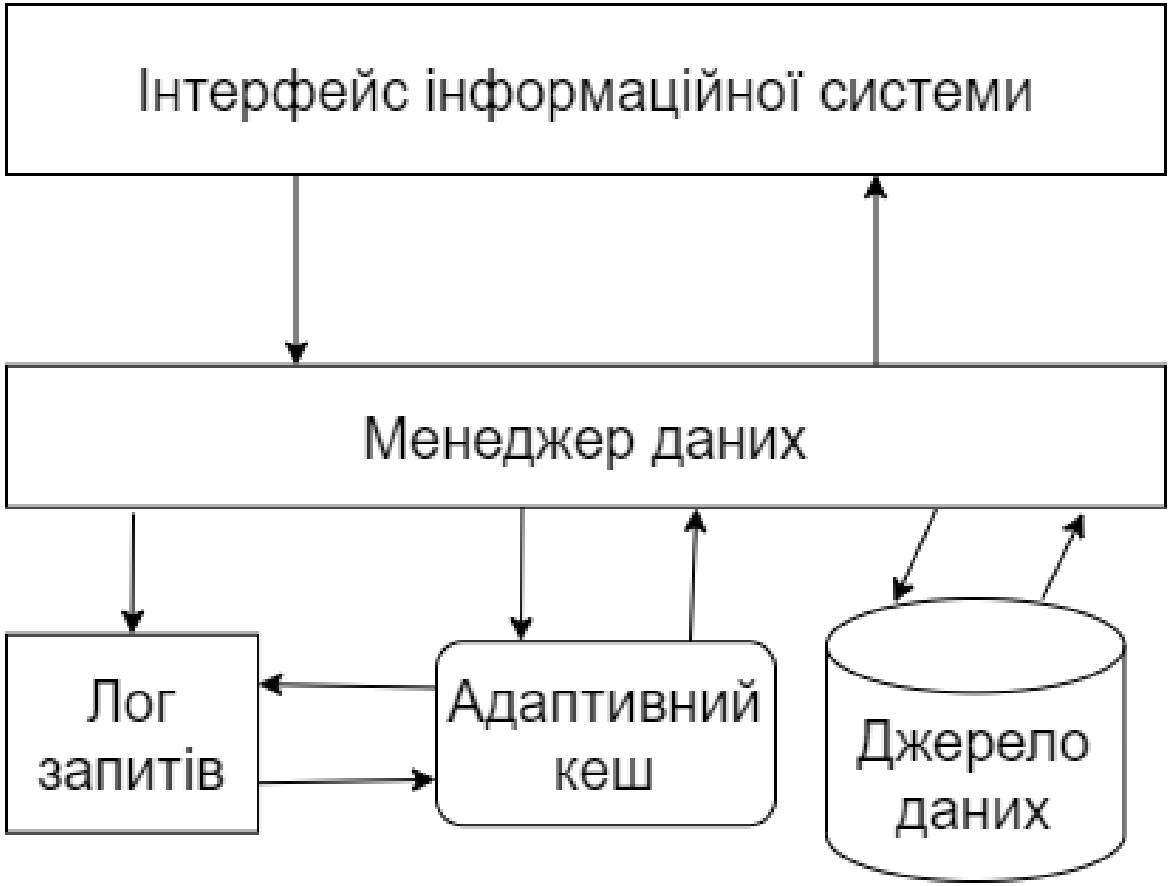
Апріорний алгоритм побудови асоціативних правил

Для побудови асоціативних правил Apriory алгоритм використовує твердження:

$$\text{Якщо } X \subseteq Y \text{ то } \sup(X) \geq \sup(Y)$$

Відповідно із розглядання видаляються усі “претенденти”, що мають підмножину, що не задовольняє граничному значенню support.

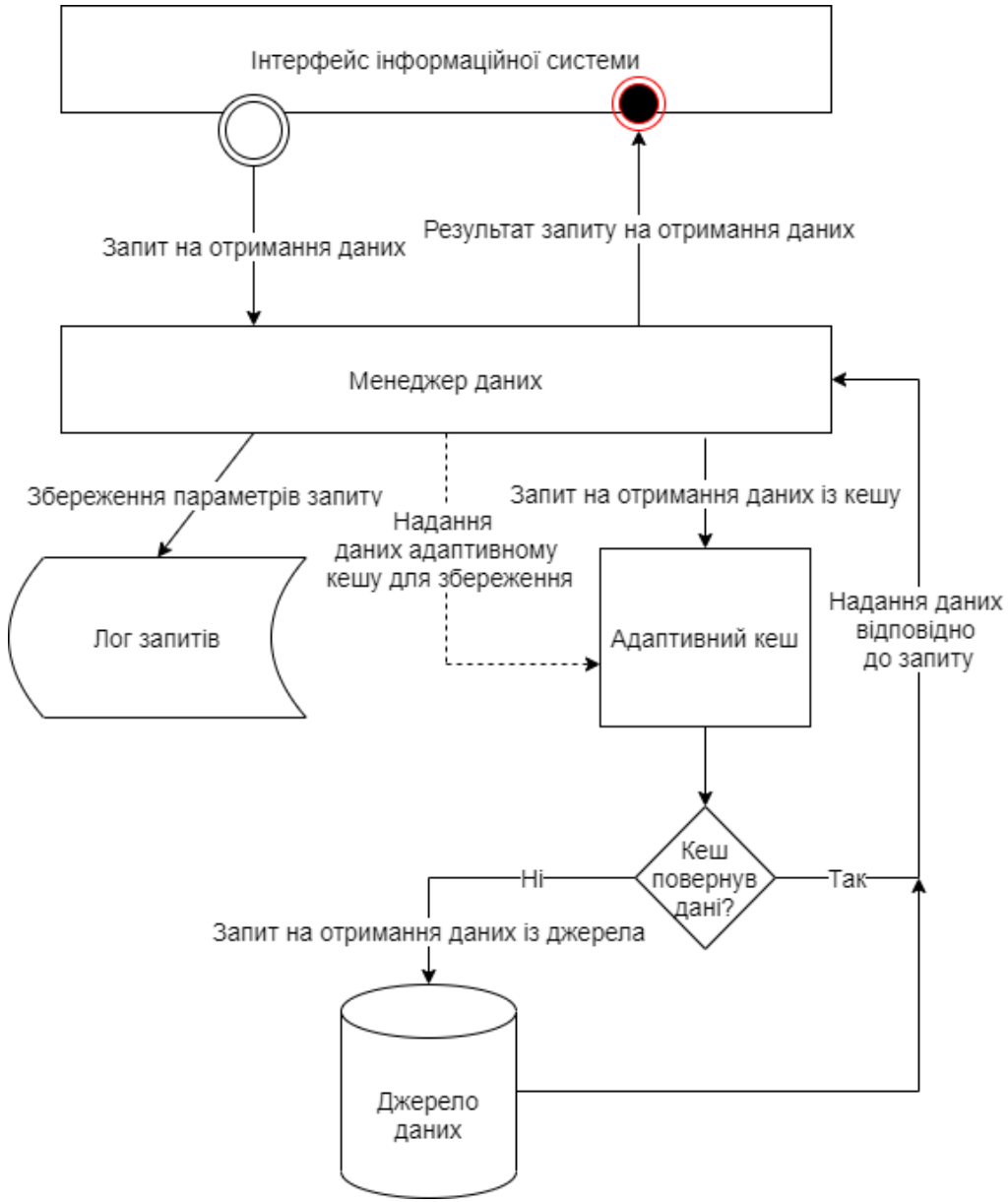
Інформаційна система



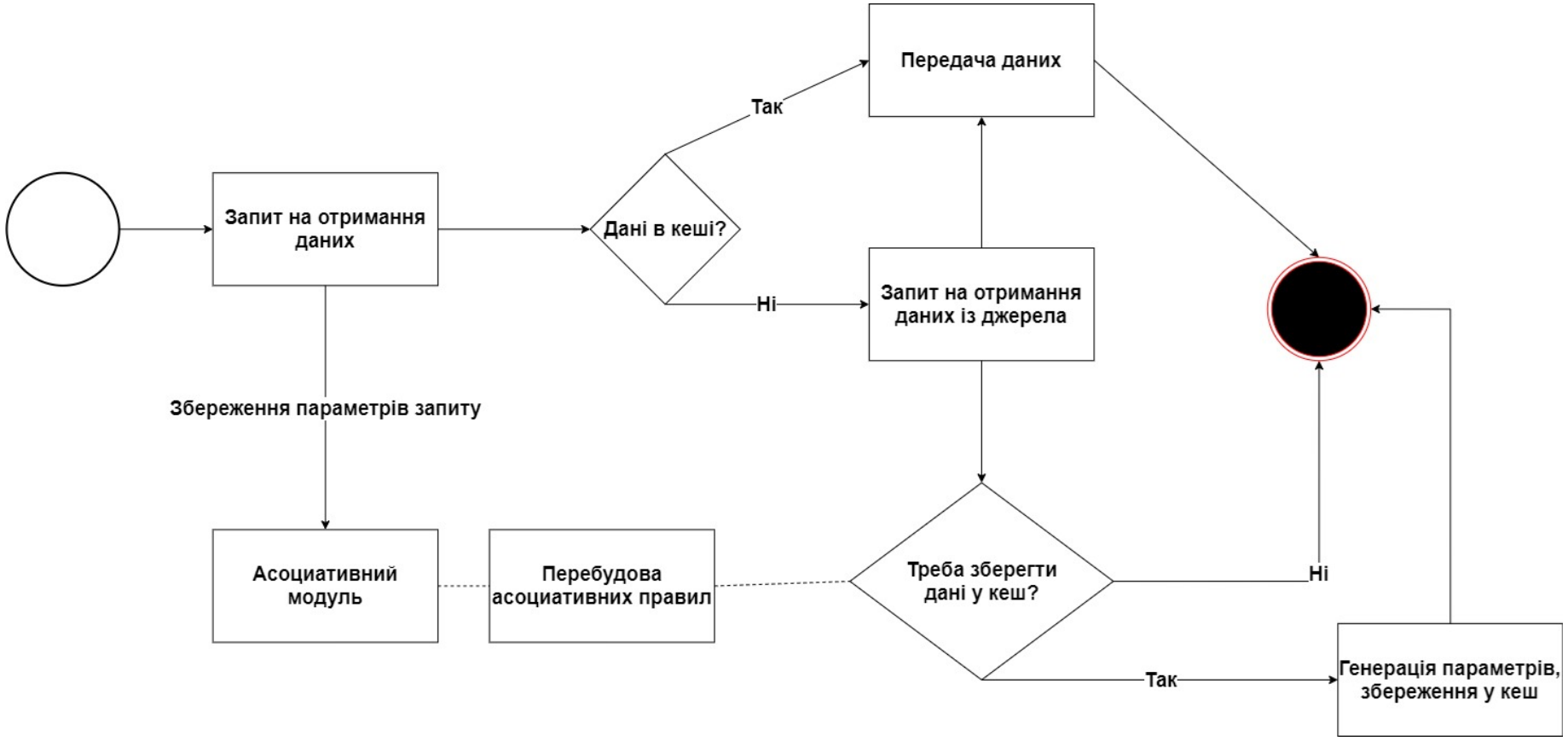
Використання асоціативних правил

- Перевірка необхідності додавання даних до кешу за асоціативними правилами
- Превентивне додавання до кешу даних, що будуть використані з великою вірогідністю у найближчий час

Алгоритм роботи інформаційної системи із адаптивним асоціативним модулем кешування



Розроблений метод кешування



Порівняння із існуючими методами

Метод кешування	Ефективність при випадкових запитах	Ефективність при наявності послідовностей у запитах	Простота реалізації	Додаткове навантаження системи	Можливості налаштування
FIFO	3	1	10	1	1
LRU	3	5	8	4	4
Адаптивний асоціативний кеш	3	9	2	5	10
Тестові дані	100 хаотичних запитів	100 запитів із залежностями			

Наукова новизна

Створений новий метод кешування. Особливостями розробленого методу кешування є адаптація системи кешування відповідно до асоціативних правил, які будуються на основі статистичних даних щодо використання інформаційної системи. Таким чином кешування інформаційної системи адаптується до попереднього використання інформаційної системи. При наявності набору таких асоціативних правил, інформаційна система буде мати можливість розміщувати результати виконання ряду запитів в кеш ще до їх виникнення. Це надає можливості системі покращити показники швидкодії надання результатів обробки запитів.

Перспективи модифікації

- Розробка алгоритмів для розбиття логу запитів на транзакції
- Розробка модулю, що буде налаштовувати роботу адаптивного асоціативного модулю кешування під характеристики системи

Висновок

Побудована модель адаптивного асоціативного кешу у інформаційній системі. Модель будується на основі побудови асоціативних правил із статистичних даних, отриманих під час роботи системи.

Програмно реалізовано модель інформаційної системи для тестування ефективності нового методу кешування

Новий метод кешування дозволяє підвищити ефективність роботи системи за допомогою адаптації кешування на основі асоціативних праваил, які будуються відповідно до використання інформаційної системи.

Дякую за увагу